

АСТРА ИИ

Платформа АСТРА ИИ

(ASTRA AI Platform)

Руководство администратора

Версия 1.0.1

Москва
2026

СОДЕРЖАНИЕ

ЧАСТЬ 1. МОДУЛЬ «ГЕНЕРАТИВНЫЙ ИИ»	13
1. Общие положения	13
1.1. Основные понятия и определения.....	13
2. О модуле «Генеративный ИИ».....	21
2.1 Разработка приложений и прототипирование	21
2.2 Модификации и интеграции	21
3. Быстрый старт.....	23
3.1 Требования	23
3.2 Запуск шаблона Simple Agent.....	23
3.3 Запуск потоков из внешних приложений	25
4. Учебные примеры.....	31
4.1 Создание чат-бота с векторным RAG	31
4.2 Создание чат-бота, который может принимать файлы.....	34
4.3 Подключение приложений к агентам	37
4.4 Подключение к серверам MCP из приложения	40
5. Визуальный редактор	46
5.1 Рабочее пространство	46
5.2 Playground	46
5.3 Поделиться	47
6. Создание потоков	48
6.1 Создание потока.....	48
6.2 Управление потоками в проектах	49
6.3 Хранение потоков и журналы.....	50
7. Запуск потоков.....	51
7.1 Запуск потоков через API Генеративный ИИ	51
7.2 Запуск потоков с помощью вебхуков	57
8. Тестирование потоков.....	61
8.1 Запуск потока в Playground	61
8.2 Пользовательские идентификаторы сессий	62
8.3 Совместный доступ к Playground потока	63
9. Импорт и экспорт	64
9.1 Экспорт потока.....	64
9.2 Импорт потока.....	64

9.3 Содержимое JSON-файла Генеративный ИИ	65
10. Использование агентов	68
10.1 Использование компонента Агент в потоке.....	68
10.2 Параметры компонента Агент	70
10.3 Вывод компонента Агент	72
11. Настройка инструментов	73
11.1 Присоединение инструментов.....	73
11.2 Использование агента в качестве инструмента	74
11.3 Добавление пользовательских компонентов в качестве инструментов	74
11.4 Сделать любой компонент инструментом.....	76
11.5 Использование потоков как инструменты	76
12. MCP клиент	77
12.1 Использование компонента «Инструменты MCP»	77
12.2 Параметры компонента «Инструменты MCP»	79
12.3 Управление подключёнными серверами MCP	80
12.4 Изменение переменных окружения сервера MCP через API	80
12.5 Использование глобальных переменных в заголовках сервера MCP.....	82
13. MCP сервер	84
13.1 Требования	84
13.2 Публикация потоков как инструментов MCP	84
13.3 Подключение клиентов к серверу MCP.....	85
13.4 Аутентификация сервера MCP	87
13.5 Переменные окружения сервера MCP	89
13.6 Тестирование и отладка с помощью MCP Inspector.....	91
13.7 Устранение неполадок.....	91
14. Astra DB MCP	92
15. Ключи API и аутентификация	94
15.1 Ключи API Генеративный ИИ.....	94
15.2 Ключи API компонентов.....	96
16. Глобальные переменные	97
16.1 Создание глобальной переменной	97
16.2 Редактирование глобальной переменной	97
16.3 Удаление глобальной переменной	98
17. Хранилище и память	99
17.1 Управление файлами	99

17.2 Управление памятью	101
17.3 Управление коммуникацией между компонентами через идентификатор сеанса ..	102
17.4 Управление векторными данными.....	103
18. Наблюдаемость	109
18.1 Журналы	109
18.2 Трассировки.....	110
19. Типы данных	112
19.1 Данные	112
19.2 DataFrame.....	113
19.3 Эмбединги	114
19.4 LanguageModel	114
19.5 Память.....	114
19.6 Сообщения.....	115
19.7 Инструмент.....	116
19.8 Неизвестные или множественные типы	116
19.9 Просмотр типов данных в потоках	117
20. Голосовой режим.....	119
20.1 Тестирование голосового режима в Playground.....	119
23. Безопасность	120
23.1 Безопасность собственных развёртываний	120
23.2 Безопасность сторонних развёртываний	120
23.3 Бюллетень безопасности	121
24. Обзор компонентов	122
24.1 Добавление компонента в поток	122
24.2 Порты компонента	123
24.3 Код компонента.....	124
24.4 Версии компонентов.....	125
24.5 Группировка компонентов.....	125
24.6 Устаревшие компоненты.....	126
25. Основные компоненты.....	127
25.1 Ввод и вывод	127
25.2 Обработка данных	132
25.3 Источник данных	147
25.4 Файлы и данные	157
25.5 Управление потоком.....	164

25.6 LLM операторы	168
25.7 Модели и агенты	182
25.8 Утилиты	201
25.8.5 Устаревшие основные компоненты	209
26. Бандлы (интеграции)	218
26.1 О пакетах	218
26.2 Agentics	226
26.3 AI/ML API.....	230
26.4 ALTK.....	232
26.5 Amazon.....	234
26.6 Anthropic	239
26.7 Apify	240
26.8 arXiv	241
26.9 AssemblyAI	241
26.10 Azure.....	245
26.11 Baidu.....	247
26.12 Bing.....	247
26.13 Cassandra	248
26.14 Chroma.....	254
26.15 Cleanlab	257
26.16 ClickHouse.....	261
26.17 Cloudflare	264
26.18 Cohere	265
26.19 CometAPI	267
26.20 Composio	269
26.21 Couchbase	273
26.22 CUGA	275
26.23 DataStax.....	278
26.24 DeepSeek	300
26.25 Docling.....	301
26.26 DuckDuckGo	304
26.27 Elastic.....	304
26.28 Exa	309
26.29 FAISS.....	310
26.30 Glean.....	311

26.31 Компоненты Google	312
26.32 Groq	316
26.33 Hugging Face	317
26.34 IBM	319
26.35 Icosa Computing	323
26.36 LangChain	323
26.37 LiteLLM	329
26.38 LM Studio	331
26.39 MariTalk	333
26.40 Mem0	333
26.41 Milvus	335
26.42 MistralAI	337
26.43 MongoDB	339
26.44 Notion	341
26.45 Novita	346
26.46 NVIDIA	347
26.47 Ollama	354
26.48 OpenAI	355
26.49 OpenRouter	358
26.50 Perplexity	359
26.51 pgvector	360
26.52 Pinecone	362
26.53 Qdrant	363
26.54 Redis	365
26.55 SambaNova	367
26.56 Serper	368
26.57 Supabase	368
26.58 Upstash	370
26.59 vLLM	371
26.60 Vectara	373
26.61 Vertex AI	375
26.62 Weaviate	378
26.63 Wikipedia	379
26.64 xAI	380
27. Начало работы с API	382

27.1	Формирование запросов к API Генеративный ИИ	382
27.2	Настройка переменных окружения	383
27.3	Попробуйте запросы к API Генеративный ИИ	384
27.4	Доступные конечные точки	385
28.	TypeScript клиент	391
28.1	Установка пакета	391
28.2	Инициализация клиента	391
28.3	Подключение к серверу и получение ответов	391
28.4	Расширенные возможности клиента	392
28.5	Получение журналов с помощью клиента TypeScript	394
29.	Триггеры потока	397
29.1	Запуск потока	397
29.2	Вебхук запуска потока	405
29.3	Устаревшие конечные точки запуска потоков	405
30.	API разработчика (Beta)	406
30.1	Workflow API (Beta)	406
31.	OpenAI Responses API	415
31.1	Предварительные требования	415
31.2	Пример запроса	416
31.3	Пример ответа	417
31.4	Пример запроса с потоковой передачей	419
31.5	Продолжение бесед с previous_response_id	420
31.6	Получение результатов вызова инструмента	421
31.7	Передача глобальных переменных через заголовки	422
31.8	Отслеживание использования токенов	422
32.	Управление потоком	423
32.1	Создать поток	423
32.2	Создать потоки	423
32.3	Прочитать поток	424
32.4	Прочитать потоки	425
32.5	Прочитать образцы потоков	425
32.6	Обновить поток	425
32.7	Удалить поток	426
32.8	Экспортировать потоки	426
32.9	Импортировать потоки	426

33. Файлы	428
33.1 Различия между /v1/files и /v2/files	428
33.2 Конечные точки файлов (v1)	428
33.3 Конечные точки файлов (v2)	430
33.4 Создание загрузки файла (Устарело)	432
34. Проекты	433
34.1 Чтение проектов	433
34.2 Создание проекта	433
34.3 Чтение проекта	434
34.4 Обновление проекта	434
34.5 Удаление проекта	434
34.6 Экспорт проекта	435
34.7 Импорт проекта	435
35. Журналы	436
35.1 Поточковая передача логи	436
35.2 Получить логи с необязательными параметрами	436
36. Мониторинг	438
36.1 Сборки Vertex	438
36.2 Конечные точки сообщений	445
36.3 Получить трассировки	447
36.4 Получить транзакции	449
37. Пользователи	450
37.1 Добавить пользователя	450
37.2 Получить текущего пользователя	450
37.3 Список всех пользователей	451
37.4 Обновить пользователя	452
37.5 Сбросить пароль	452
37.6 Удалить пользователя	453
38. Устранение неполадок	454
38.1 Нет поля ввода в Playground	454
38.2 Отсутствует ключ, ключ не найден или неверный API-ключ	454
38.3 Проблемы MCR в модуле «Генеративный ИИ»	454
38.4 Ошибки ограничения длины токенов в компонентах модели эмбедингов	456
ЧАСТЬ 2. МОДУЛЬ «МАШИННОЕ ОБУЧЕНИЕ»	457
39. Введение	458

39.1. Область применения.....	458
40. Руководство администратора платформы Астра ИИ.....	459
40.1 Начало работы.....	459
40.2 Проекты	459
41. Начало работы с платформой.....	460
41.1 Авторизация	460
41.2 Регистрация	460
42. Обзор интерфейса.....	462
42.1 Файловое хранилище.....	462
42.2 Уведомления.....	465
42.3 Личный кабинет	465
43. Управление проектами.....	469
43.1 Как создать проект.....	469
43.2 Работа с существующими проектами	470
44. Меню проекта	472
45. Карточка проекта.....	474
45.1 Датасеты	475
45.2 Модели ИИ	475
45.3 Коннекторы	476
45.4 Дашборды	476
46. Как создать датасет на основе загруженных данных.....	477
46.1 Создать датасет из загруженных файлов.....	477
46.2 Управление датасетами.....	480
46.2.1. Экспорт датасета.....	480
46.2.2. Импорт датасета	481
46.3 Как создать датасет из источника	482
Шаг 1: Выбор базы данных	482
Шаг 2: Формирование SQL-запроса	483
Шаг 3: Выполнение запроса и предварительный просмотр.....	484
Шаг 4: Сохранение результата в датасет	485
Шаг 5: Проверка	485
47. Обучение модели ИИ	486
47.1 Начало работы: создаем рабочую область	486
47.2 Управление рабочими областями.....	487
47.2.1. Экспорт рабочей области.....	487
47.2.2. Импорт рабочей области.....	487
47.3 Обзор рабочей области.....	488

47.3.1. Панель добавления блоков	488
47.3.2 Фильтры.....	489
47.4 Процесс сборки пайплайна	491
47.4.1. Добавление и перемещение блоков	491
47.4.2. Настройки блока	492
47.4.3. Соединение блоков.....	493
47.4.4. Удаление связей.....	493
47.4.5. Визуализация	494
48. Практические примеры. Разведочный анализ данных.....	496
48.1 Загрузка данных.....	496
Настройка	496
48.2 Анализ корреляций	498
48.2.1 Анализ результатов	498
48.3 Поиск пропущенных значений.....	499
48.3.1. Анализ результатов	499
48.4 Анализ PCA.....	500
Настройка	500
48.4.1. Анализ результатов	501
1. По тепловой карте коэффициентов:	501
2. По 3D-графику распределения:.....	501
48.5 Визуальный анализ распределений.....	502
48.5.1. Анализ результатов	502
48.5.2. Столбчатая диаграмма (для категориальных признаков).....	502
48.6 Расчет описательных статистик	504
11.6.1. Анализ результатов	504
49. Примеры обучения моделей	506
49.1 Классификация. Определение вида животного	506
Шаг 1. Загрузка данных	506
Шаг 2. Выбор признаков.....	507
Шаг 3. Разделение выборки.....	508
Шаг 4. Кодирование целевой переменной	508
Шаг 5. Обучение модели	509
Шаг 6. Сохранение модели.....	510
Шаг 7. Валидация модели.....	510
49.2 Анализ результатов.....	511
1. Матрица ошибок (Confusion Matrix).....	511
2. График кривой ROC-AUC	511
3. Таблица метрик.....	511
4. Классификационный отчет (Classification Report)	511
49.3 Регрессия. Предсказание возраста животного	513
Шаг 1. Загрузка данных	514
Шаг 2. Выбор признаков.....	515

Шаг 3. Разделение выборки.....	516
Шаг 4. Бинарное кодирование.....	516
Шаг 5. Обучение модели	517
Анализ результата обучения	517
Шаг 6. Сохранение модели.....	518
Шаг 7. Валидация модели.....	518
Анализ результатов валидации	519
49.4Кластеризация. Поиск скрытых групп.....	520
Шаг 1. Загрузка данных	520
Шаг 2. Выбор признаков.....	522
Шаг 3. Подготовка данных: Кодирование	522
Шаг 4. Обучение модели. Поиск кластеров.....	523
Шаг 5. Оценка качества кластеризации	524
50. Примеры сложных пайплайнов.....	527
50.1Классификация. Определение источника сообщения.....	527
Шаг 1. Загрузка данных	527
Шаг 2. Выбор признаков.....	528
Шаг 3. Разделение выборки.....	529
Шаг 4. Кодирование целевой переменной	529
Шаг 5. Предобработка текста: Токенизация	530
Шаг 6. Предобработка текста: Лемматизация	530
Шаг 7. Векторизация текста	532
Шаг 8. Параллельное обучение трех моделей	532
Шаг 9. Валидация и сравнение моделей	534
50.2Анализ результатов валидации.....	535
50.3 Регрессия. Улучшение модели с помощью инженерии признаков	535
Шаг 1. Загрузка данных	536
Шаг 2. Создание агрегированных признаков	537
Шаг 3. Округление значений.....	538
Шаг 4. Выбор новых признаков	539
Шаг 5. Обогащение датасета	540
Шаг 6. Параллельная подготовка данных	542
Шаг 7. Разделение выборки.....	543
Шаг 8. Кодирование признаков.....	543
Шаг 9. Обучение моделей.....	543
Шаг 10. Валидация и сравнение моделей	545
50.4 Анализ результатов валидации.....	546
50.5 Кластеризация. Поиск групп с помощью агрегированных признаков	547
Шаг 1. Загрузка данных	548
Шаг 2. Поиск пропущенных значений	549
Шаг 3. Удаление пропусков	550
Шаг 4. Агрегация.....	551
Шаг 5. Округление и модификация значений	552

Шаг 6. Выбор столбцов.....	552
Шаг 7. Объединение датасетов	553
Шаг 8. Округление и модификация значений	553
Шаг 9. Параллельная подготовка данных	554
Шаг 10. Кодирование признаков.....	555
Шаг 11. Стандартизация	556
Шаг 12. Обучение моделей и анализ результатов.....	556
51. Использование модели ИИ	559
51.1 Управление рабочими областями.....	559
51.2 Обзор рабочей области.....	559
51.2.1. Панель добавления блоков	560
51.2.2. Процесс сборки пайплайна.....	560
52. Примеры использования моделей	561
52.1 Применение модели классификации	561
Шаг 1. Загрузка данных	561
Шаг 2. Загрузка обученной модели	562
Шаг 3. Применение модели классификации и получение прогнозов	563
52.2 Применение модели регрессии.....	564
Шаг 1. Загрузка данных	565
Шаг 2. Загрузка модели.....	566
Шаг 3. Применение модели регрессии и получение прогнозов	567
52.3 Применение модели кластеризации.....	568
Шаг 1. Загрузка новых данных	568
Шаг 2. Загрузка обученной модели	569
Шаг 3. Применение модели и анализ результатов	570
53. Порядок работы с коннекторами	572
54. Порядок работы с дашбордами	574
54.1 Управление виджетами на рабочей области	574
54.2 Выбор коннектора для виджета.....	575
54.3 Настройка расположения	576

ЧАСТЬ 1. МОДУЛЬ «ГЕНЕРАТИВНЫЙ ИИ»

1. Общие положения

1.1. Основные понятия и определения

Таблица 1 – Основные термины, используемые в документе, и их определения

Термин	Определение
Apache Kafka	Брокер сообщений, реализующий паттерн Producer-Consumer. Данные из одного и того же топика могут считываться множеством консьюмер-групп одновременно.
Apache Spark	Фреймворк с открытым исходным кодом для реализации распределенной обработки неструктурированных и слабоструктурированных данных.
BPMN	С англ. Business Process Model and Notation. Нотация, определяющая способ визуализации процессов в виде диаграмм с определенным набором блоков и взаимосвязей.
YOLO	С англ. You Only Look Once. Архитектура нейронных сетей, предназначенная для <i>детекции объектов</i> на изображении. Отличительной особенностью является подход к решению задачи детекции: исходное изображение сжимается таким образом, чтобы получить <i>квадратную матрицу</i> размером 13 на 13, в каждой клетке которой записана информация о наличии объекта и классе этого объекта на соответствующей части картинке. Таким образом, YOLO просматривает картинку один раз, что существенно увеличивает скорость обработки.
YOLOv5	Усовершенствованная пятая версия YOLO, реализованная на фреймворке PyTorch.
Временной ряд	Совокупность наблюдений, собранных за определенный временной интервал. Этот тип данных используется для поиска долгосрочного тренда, прогнозирования будущего и прочих видов анализа. Анализ временных рядов позволяет обнаруживать тенденции и закономерности в исследуемых процессах, строить прогнозы и предсказывать будущие изменения в бизнесе, на производстве, и в других областях.
Выборка	Случайное подмножество генеральной совокупности.
Датасет	С англ. <i>Data set</i> , набор данных. Коллекция из логических записей, хранящихся в виде <i>кортежа</i> . Набор данных можно сравнить с файлом, но в отличие от файла он является одновременно и каталогом, и

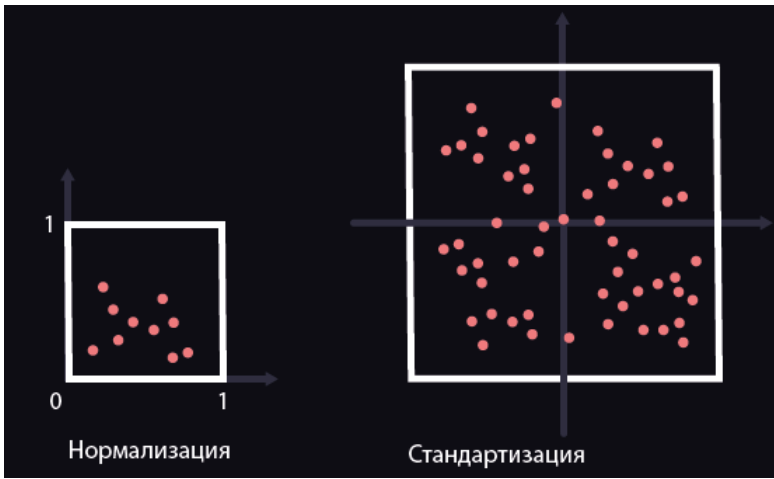
	файлом файловой системы, и не может содержать в себе другие наборы. Файловая система ориентирована на хранение записей, которые являются неделимыми единицами хранения. Множества записей объединяются в группы, которые и называются наборами данных. Записи в наборах данных используются приложениями, например, как входные данные. Так, записями набора данных могут быть как текстовые данные, так и изображения. К набору данных можно обратиться, указав точное место его хранения, или, если ранее для набора было зазервировано имя в файловой системе, по имени (второй вариант не реализован). Также <i>датасетом</i> называются данные, которые пользователь загрузил с локального компьютера, а Система при загрузке выполнила их предварительную обработку. Например, в систему загружается временной ряд в формате csv, и при загрузке он преобразуется в <i>структурированные данные</i> или по-другому датасет. Такой датасет пригоден для использования в моделях машинного обучения. В качестве <i>датасета</i> в настоящем документе также выступает набор изображений или файлов, которые используются для решения одной задачи.
Дерево решений	Инструмент прогнозного моделирования. Строится с помощью алгоритмического подхода, который разделяет набор данных на основе различных условий. Относится к классу обучения с учителем, используется для задач классификации и регрессии. Цель в том, чтобы создать модель, которая предсказывает значение целевой переменной, изучая правила принятия решений, выведенные из характеристик данных. Алгоритм: 1. Выбрать лучший атрибут, который разделяет наблюдения на группы. 2. Задать соответствующий вопрос. 3. Следовать по путям ответов. 4. Вернуться к шагу 1.
Дисперсия	Мера удаленности того или иного значения выборки от среднего значения. Рассчитывается по формуле: $\sigma^2 = \frac{\sum (x_i - \bar{x})^2}{N},$ где σ^2 – дисперсия случайной величины, x_i – i-ый элемент выборки, $\bar{x}$ – среднее арифметическое, N – количество элементов выборки.

	Стандартное отклонение рассчитывается как квадратный корень из полученной цифры.
Интеллектуальный анализ текстов	Направление в искусственном интеллекте, целью которого является получение информации из коллекции текстовых документов, основываясь на применении эффективных в практическом плане методов машинного обучения и обработки естественного языка. Ключевыми группами задач ИАТ являются: категоризация текстов, извлечение информации и информационный поиск, изменение информации в коллекциях текстов. Категоризация документов заключается в отнесении документов из коллекции к одной или нескольким группам (классам, кластерам) схожих между собой текстов. Категоризация с участием человека называется <i>классификацией документов</i>, система ИАТ должна отнести тексты к уже определенным классам. Для этого производится обучение с учителем, для чего пользователь предоставляет системе ИАТ как множество классов, так и образцы документов, принадлежащих этим классам. Второй случай категоризации называется <i>кластеризацией документов</i>. При этом система ИАТ сама определяет множество кластеров, по которым могут быть распределены тексты (производится обучение без учителя). В этом случае пользователь сообщает системе ИАТ количество кластеров, на которое ему хотелось бы разбить обрабатываемую коллекцию (в алгоритме программы уже заложена процедура выбора признаков).
Категориальная переменная (качественные данные)	Это данные с ограниченным числом уникальных значений или <i>категорий</i> (например, пол, страна проживания, номер группы, категория товаров, и т.п.). Категориальные поля могут быть как текстовыми, так и числовыми, в которых категории закодированы <i>числовыми кодами</i> (например, 0=женский, а 1=мужской). Номинальные поля, порядковые поля и флаги являются категориальными полями. -<i>Набор</i> (номинальная переменная). Поле, значения которого представляют категории без естественного упорядочивания (например, подразделение компании, в котором работает сотрудник). -<i>Упорядоченный набор</i> (порядковая переменная). Поле, значения которого представляют категории с некоторым естественным для них упорядочением (например, оценки, представляющие степень удовлетворенности или уверенности, или баллы, оценивающие предпочтение). -<i>Флаг</i>. Поле или переменная с двумя отдельными значениями, например Да и Нет.

Классификация	Задача машинного обучения, которая ставит своей целью назначить метку класса наблюдениям из предметной области. Основные типы классификации: — бинарная классификация; — мультиклассовая классификация; — классификация по нескольким меткам; — несбалансированная классификация.
Кластеризация	Техника обучения без учителя, которая включает в себя группирование или кластеризацию точек данных. Чаще всего она используется для сегментации потребителей, выявления мошенничества, классификации документов и определении ключевых слов. Кластеризация (или кластерный анализ) – это задача разбиения множества объектов на группы, называемые кластерами. Главное отличие от классификации состоит в том, что перечень групп четко не задан и определяется в процессе работы алгоритма.
Машинное обучение (ML – Machine learning)	Тренировка математической модели на исторических данных для того, чтобы прогнозировать какое-то событие или явление на новых данных. То есть попытка заставить алгоритмы программ совершать действия на основе предыдущего опыта, а не только на основе имеющихся данных. Для обучения нужны исторические данные (обучающая выборка) и значение целевой переменной (то, что прогнозируем), которое соответствует заданным историческим данным. Модель наблюдает и находит зависимости между данными и целевой переменной. Эти зависимости используются моделью для нового набора данных, чтобы прогнозировать целевую переменную, которая неизвестна. Машинное обучение включает в себя целый набор методов и алгоритмов, которые могут предсказать какой-то результат по входным данным. Алгоритмов машинного обучения большое множество: одни эффективны для решения одного типа задач, вторые – для другого. Суть технологии машинного обучения Говоря в общем, машинное обучение – это обучение компьютерной программы или алгоритма постепенному улучшению исполнения поставленной задачи. Машинное обучение обозначает множество математических, статистических и вычислительных методов для разработки алгоритмов, способных решить задачу не прямым способом, а на основе поиска закономерностей в разнообразных входных данных.

	Решение вычисляется не по четкой формуле, а по установленной зависимости результатов от конкретного набора признаков и их значений. Например, если каждый день в течение недели земля покрыта снегом и температура воздуха существенно ниже нуля, то вероятнее всего, наступила зима. Поэтому машинное обучение применяется для диагностики, прогнозирования, распознавания и принятия решений в различных прикладных сферах: от медицины до банковской деятельности.
Мониторинг	Процесс наблюдения и регистрации данных о каком-либо объекте на неразрывно примыкающих друг к другу интервалах времени, в течение которых значения данных существенно не изменяются.
Мониторинг состояния	Наблюдение за состоянием объекта для определения и предсказания момента перехода в предельное состояние. Результат мониторинга состояния объекта представляет собой совокупность диагнозов составляющих его субъектов, получаемых на неразрывно примыкающих друг к другу интервалах времени, в течение которых состояние объекта существенно не изменяется. Принципиальное отличие от мониторинга параметров является наличие интерпретатора измеренных параметров в терминах состояния – экспертной системы поддержки принятия решений о состоянии объекта и дальнейшем управлении.
Наблюдение (строка, запись, точка, сущность)	Ценные данные, собираемые во время исследования или эксперимента. Вместе с масштабом анализа определяет совокупность. <i>Эмпирические исследования</i> – практические эксперименты с результатами на основе реального опыта, а не теории или убеждений. Основопологающим принципом <i>Науки о данных</i> является приоритет наблюдения над предположением. Типы наблюдений: — <i>Числовой</i>: целые (integer), вещественные (real number), числа с плавающей точкой (float). — <i>Булевый</i> (boolean) – принимает значения 1/0 (да/нет). — <i>Категориальный</i>. Например, жанры кино: комедия, ужасы, мелодрама. — <i>Текстовый</i>. — <i>Вектор</i>.
Нейронная сеть (или Искусственная нейронная сеть)	Представляет собой <i>математическую модель</i> , а также её программное или аппаратное воплощение, построенную по принципу организации и функционирования биологических нейронных сетей – сетей нервных клеток живого организма.

	Нейронные сети решают задачу: по точкам находят функцию. Происходит это путем <i>минимизации ошибки</i> – сводится к минимуму «расстояние» между значениями, предсказываемыми нейронной сетью, и значениями, которые наблюдаются. Под <i>архитектурой нейронной сети</i> понимается ее устройство – последовательность нейронов и связей между ними.
Нормализация	Техника преобразования значений признака, масштабирующая значения таким образом, что они расположены в диапазоне от 0 до 1.
Обучение с учителем	Контролируемое обучение – метод машинного обучения, при котором модель обучается на размеченных данных. Например, исследовав опухоли, установив их размер, плотность и другие метрики, мы передаем эти данные модели с обязательной пометкой, какое наблюдение к какому строению (доброкачественному или злокачественному) относится. Алгоритмы контролируемого обучения подразделяются на следующие модели: классификация, регрессия.
Пайплайн	Последовательность стадий, внутри которых расположены задачи. Расположены они таким образом, что выход каждого элемента является входом следующего.
Признак	Объективная характеристика, характерная черта или свойство, которое может быть определено или измерено. В статистике независимые переменные X используются для предсказания зависимого признака Y
Регрессия математической статистике)	(в Математическое выражение, отражающее связь между зависимой переменной y и независимыми переменными x. Алгоритмы регрессии используются для контролируемого обучения моделей искусственного интеллекта. Модели обучают прогнозировать числовые значения целевых переменных.

Стандартизация	Техника преобразования значений признака, адаптирующая признаки с разными диапазонами значений к моделям машинного обучения, использующих дистанцию для прогнозирования. Это разновидность нормализации с использованием стандартизированной оценки преобразует значения так, что из каждого наблюдения каждого признака вычитается среднее значение и результат делится на стандартное отклонение этого признака: $x_{i,\text{станд.}} = \frac{x_i - \mu}{\sigma},$ где $x_{i,\text{станд.}}$ – стандартизированный элемент признака, x_i – исходный элемент, μ – среднее арифметическое, σ – стандартное отклонение. Преобразование необходимо, поскольку признаки датасета могут иметь большие различия между своими диапазонами, и для моделей машинного обучения это спровоцирует искаженное восприятие данных. Стандартизация, в отличие от нормализации, не имеет ограничивающего диапазона: 
Стандартизированная оценка	Метрика, характеризующая удаленность наблюдения от среднего значения совокупности данных.
Стандартное отклонение	Мера разброса в наборе числовых данных. Показывает, насколько далеко от среднего арифметического находятся точки данных. Чем меньше стандартное отклонение, тем более сгруппированы данные вокруг центра (среднего). Чем отклонение больше, тем больше разброс значений.
Тренировочные (обучающие) данные	Часть датасета, обучающая основа модели машинного обучения. Является одной из составляющих разделенного набора данных наряду с <i>тестовыми</i> и <i>валидационными</i> данными.

Валидационные (тестовые) данные	Часть датасета, основа для проверки работоспособности модели машинного обучения.
Числовая переменная	Numeric variable - переменная, выраженная различными видами чисел.
Целевая (зависимая) переменная	Target variable – признак датасета, который предстоит предсказывать модели машинного обучения. Зависимой ее называют, поскольку в ходе разведочного анализа данных выявляется корреляция между одной или несколькими переменными-предикторами и рассматриваемым целевым признаком.

2. О модуле «Генеративный ИИ»

Генеративный ИИ — это открытый, написанный на Python настраиваемый фреймворк для создания AI-приложений. Он поддерживает ключевые функции AI, такие как агенты и протокол Model Context Protocol (MCP), и не привязан к конкретным большим языковым моделям (LLM) или векторным хранилищам.

Визуальный редактор упрощает прототипирование рабочих процессов, позволяя разработчикам быстро превращать идеи в полноценные решения.

Создайте и запустите первый поток за несколько минут: установите Генеративный ИИ, затем пройдите быстрый старт.

2.1 Разработка приложений и прототипирование

Генеративный ИИ помогает разрабатывать широкий спектр AI-приложений: чат-боты, системы анализа документов, генераторы контента и агентные приложения.

Генеративный ИИ включает готовые шаблоны, которые можно использовать сразу или настроить под свои нужды.

Создавайте потоки за несколько минут

Основная задача Генеративный ИИ — создавать и обслуживать потоки, представляющие собой функциональные описания рабочих процессов приложений.

Чтобы создать поток, соедините и настройте узлы компонентов. Каждый компонент — отдельный шаг в рабочем процессе.

С помощью визуального редактора вы можете перетаскивать компоненты и быстро создавать, и тестировать рабочие процессы AI-приложений. Например, можно создать поток чат-бота для интернет-магазина, который использует LLM и хранилище данных о продуктах и позволяет клиентам задавать вопросы о товарах.

Тестирование потоков в реальном времени

Используйте **Playground** для тестирования потоков без развёртывания всего стека приложения. Вы можете взаимодействовать с потоками и получать обратную связь в реальном времени о логике выполнения и генерации ответов. Также можно запускать отдельные компоненты для тестирования зависимостей в изоляции.

Запуск и обслуживание потоков

Потоки можно использовать как прототипы для дальнейшей разработки приложений либо встраивать в код своего приложения через API Генеративный ИИ.

Для более масштабной разработки вы можете подключить Генеративный ИИ как зависимость или развернуть сервер Генеративный ИИ для обслуживания потоков через публичный интернет.

Дополнительная информация:

- Запуск потоков через API Генеративный ИИ
- Контейнеризация приложения Генеративный ИИ

2.2 Модификации и интеграции

Генеративный ИИ предоставляет компоненты, поддерживающие множество сервисов, инструментов и функций, необходимых для AI-приложений.

Одни компоненты являются универсальными — например, ввод данных, вывод данных и хранилища данных. Другие — специализированными: агенты, языковые модели и провайдеры эмбедингов.

Все компоненты предоставляют параметры, которым можно задавать фиксированные или переменные значения. Для временного переопределения параметров потока во время выполнения используйте tweaks.

Поддержка агентов и МСР

Помимо создания агентных потоков, вы можете использовать встроенные функции агентов и МСР в модуле «Генеративный ИИ»:

- Использовать Генеративный ИИ Agent
- Использовать компоненты и потоки в качестве инструментов агента
- Использовать Генеративный ИИ в качестве сервера МСР
- Использовать Генеративный ИИ в качестве клиента МСР

Расширяемость

Помимо основных компонентов, Генеративный ИИ поддерживает пользовательские компоненты.

Вы можете использовать компоненты, созданные другими разработчиками, а также создавать собственные — для личного использования или публикации для других пользователей Генеративный ИИ.

Дополнительная информация:

- Внести вклад в модуль «Генеративный ИИ»
- Создать пользовательские Python-компоненты
- Получить помощь и запросить улучшения

3. Быстрый старт

Начните работу с модулем «Генеративный ИИ», загрузив шаблонный поток, запустив его, а затем обратившись к нему через конечную точку API /run.

3.1 Требования

- Установите и запустите Генеративный ИИ
- Создайте ключ API OpenAI
- Создайте ключ API Генеративный ИИ

Как создать ключ API Генеративный ИИ

Ключ API Генеративный ИИ — это токен, привязанный к пользователю, который используется для аутентификации запросов к модулю «Генеративный ИИ».

Чтобы создать ключ API Генеративный ИИ:

1. В Генеративный ИИ нажмите на значок вашего пользователя и выберите **Settings**.
2. Нажмите **Генеративный ИИ API Keys**, затем **Add New**.
3. Введите название ключа и нажмите **Create API Key**.
4. Скопируйте ключ API и сохраните его в надёжном месте.
5. Чтобы использовать ключ API в запросе, задайте переменную окружения `ASTRA_MULTIAGENT_AI_API_KEY` в терминале и включите заголовок `x-api-key` в ваш запрос:

Установить переменную

```
export ASTRA_MULTIAGENT_AI_API_KEY="sk..."
```

Отправить запрос

```
curl --request POST \
  --url "http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID" \
  --header "Content-Type: application/json" \
  --header "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \
  --data '{
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "Hello"
  }'
```

Если вы столкнулись с ошибкой «An API key must be passed as query or header» при попытке зарегистрироваться, см. раздел Устранение неполадок.

3.2 Запуск шаблона Simple Agent

1. В Генеративный ИИ нажмите **New Flow**, затем выберите шаблон **Simple Agent**.

Шаблон **Simple Agent** состоит из компонента **Agent**, подключённого к компонентам **Chat Input** и **Chat Output**, компонента **Calculator** и компонента **URL**. При запуске потока вы отправляете запрос агенту через компонент **Chat Input**, агент использует инструменты **Calculator** и **URL** для генерации ответа, а затем возвращает его через компонент **Chat Output**.

Многие компоненты могут выступать в качестве инструментов для агентов, включая серверы Model Context Protocol (MCP). Агент решает, какие инструменты вызывать, исходя из контекста запроса.

2. В компоненте **Agent** нажмите **Setup Provider**, чтобы выбрать поставщика языковой модели.

Чтобы настроить глобальную конфигурацию поставщиков моделей в модуле «Генеративный ИИ»:

1. Нажмите на значок профиля, выберите **Settings**, затем нажмите **Model Providers**.
2. На панели **Model Providers** выберите поставщика.
3. В поле **API Key** введите API-ключ вашего поставщика.
Ключ должен иметь разрешение на вызов моделей, которые вы хотите использовать, а ваш аккаунт должен иметь достаточный баланс. Для каждого поставщика можно добавить только один ключ — убедитесь, что он имеет доступ ко *всем* нужным вам моделям.
4. Включите конкретные модели, которые вы хотите использовать. Доступные модели зависят от поставщика и прав вашего API-ключа. Языковые модели перечислены в разделе **Language Models**, модели эмбедингов — в разделе **Embedding Models**.

После включения модели в глобальной конфигурации вы сможете использовать её в любом компоненте ваших потоков.

3. В компоненте **Agent** выберите настроенную модель в выпадающем списке **Language Model**.

Доступ к дополнительным моделям и поставщикам

Существует два способа получить доступ к дополнительным моделям и поставщикам:

- Отредактировать глобальную конфигурацию **Model Providers** в модуле «Генеративный ИИ». Используйте поставщика **Ollama**, чтобы подключиться к любой модели, размещённой на локальном или удалённом экземпляре Ollama.
- Подключить любой дополнительный компонент языковой модели к порту **Language Model** компонента **Agent**.

4. Чтобы запустить поток, нажмите **Playground**.
5. Чтобы протестировать инструмент **Calculator**, задайте агенту простой математический вопрос, например I want to add 4 and 4..

Playground отображает процесс рассуждения агента: как он анализирует запрос, выбирает инструмент и использует его для генерации ответа. В данном случае математический вопрос заставляет агента выбрать инструмент **Calculator** и выполнить действие, например `evaluate_expression`.

6. Чтобы протестировать инструмент **URL**, задайте агенту вопрос о текущих событиях. Агент выбирает действие `fetch_content` инструмента **URL** и возвращает сводку текущих новостных заголовков.
7. Когда тестирование завершено, нажмите **Close**.

Теперь, когда вы запустили первый поток, попробуйте:

- Отредактировать поток **Simple Agent**, присоединив различные инструменты или добавив компоненты.
- Создать собственные потоки с нуля или на основе других шаблонов.
- Интегрировать потоки в свои приложения — см. Запуск потоков из внешних приложений.

3.3 Запуск потоков из внешних приложений

Генеративный ИИ — это IDE, но также среда выполнения, которую можно вызывать через модуль «Генеративный ИИ» API с помощью Python, JavaScript или HTTP.

При локальном запуске Генеративный ИИ вы можете отправлять запросы на локальный сервер. Для производственных приложений необходимо развернуть стабильный экземпляр Генеративный ИИ для обработки API-запросов.

Генеративный ИИ предоставляет готовые фрагменты кода для начала работы с API.

1. При редактировании потока нажмите **Share**, затем **API access**.

Код в панели доступа к API формирует запрос к серверу Генеративный ИИ с url, headers и телом запроса payload. Фрагменты кода автоматически подставляют значения ASTRA_SERVER_ADDRESS и FLOW_ID, а также включают скрипт для передачи ASTRA_MULTIAGENT_AI_API_KEY, если вы задали его как переменную окружения. Замените эти значения при использовании кода для другого сервера или потока. Адрес сервера Генеративный ИИ по умолчанию: http://localhost:7860.

Python:

```
import requests
```

```
url = "http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID"
```

```
payload = {
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "hello world!"
}
```

```
headers = {
    "Content-Type": "application/json",
    "x-api-key": "$ASTRA_MULTIAGENT_AI_API_KEY"
}
```

try:

```
response = requests.request("POST", url, json=payload, headers=headers)
response.raise_for_status()
print(response.text)
```

```
except requests.exceptions.RequestException as e:
```

```
    print(f"Error making API request: {e}")
```

```
except ValueError as e:
```

```
    print(f"Error parsing response: {e}")
```

JavaScript:

```
const payload = {
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "hello world!",
    "session_id": "user_1"
};
```

```
const options = {
    method: 'POST',
    headers: {
        'Content-Type': 'application/json',
        'x-api-key': 'ASTRA_MULTIAGENT_AI_API_KEY'
    },
    body: JSON.stringify(payload)
};
```

```

fetch('http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID', options)
  .then(response => response.json())
  .then(response => console.log(response))
  .catch(err => console.error(err));

```

curl:

```

curl --request POST \
  --url 'http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID?stream=false' \
  --header 'Content-Type: application/json' \
  --header "x-api-key: ASTRA_MULTIAGENT_AI_API_KEY" \
  --data '{
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "hello world!"
  }'

```

Ответ 200 подтверждает успешный вызов.

2. Скопируйте фрагмент, вставьте его в файл скрипта и запустите скрипт для отправки запроса. Если вы используете фрагмент curl, команду можно выполнить напрямую в терминале.

При успешном запросе ответ содержит детали о запуске потока: идентификатор сессии, входные данные, выходные данные, компоненты, длительности и многое другое. Пример ответа от шаблонного потока **Simple Agent**:

```

{
  "session_id": "29deb764-af3f-4d7d-94a0-47491ed241d6",
  "outputs": [
    {
      "inputs": {
        "input_value": "hello world!"
      },
      "outputs": [
        {
          "results": {
            "message": {
              "text_key": "text",
              "data": {
                "timestamp": "2025-06-16 19:58:23 UTC",
                "sender": "Machine",
                "sender_name": "AI",
                "session_id": "29deb764-af3f-4d7d-94a0-47491ed241d6",
                "text": "Hello world! 🌐 How can I assist you today?",
                "files": [],
                "error": false,
                "edit": false,
                "properties": {
                  "text_color": "",
                  "background_color": "",
                  "edited": false,
                  "source": {
                    "id": "Agent-ZOknz",
                    "display_name": "Agent",
                    "source": "gpt-4o-mini"
                  }
                },
                "icon": "bot",
                "allow_markdown": false,
                "positive_feedback": null,
                "state": "complete",
                "targets": []
              },
              "category": "message",
              "content_blocks": [

```

```

{
  "title": "Agent Steps",
  "contents": [
    {
      "type": "text",
      "duration": 2,
      "header": { "title": "Input", "icon": "MessageSquare" },
      "text": "**Input**: hello world!"
    },
    {
      "type": "text",
      "duration": 226,
      "header": { "title": "Output", "icon": "MessageSquare" },
      "text": "Hello world! 🌐 How can I assist you today?"
    }
  ],
  "allow_markdown": true,
  "media_url": null
}
],
"id": "f3d85d9a-261c-4325-b004-95a1bf5de7ca",
"flow_id": "29deb764-af3f-4d7d-94a0-47491ed241d6",
"duration": null
}
},
"outputs": {
  "message": {
    "message": "Hello world! 🌐 How can I assist you today?",
    "type": "text"
  }
}
},
"messages": [
  {
    "message": "Hello world! 🌐 How can I assist you today?",
    "sender": "Machine",
    "sender_name": "AI",
    "session_id": "29deb764-af3f-4d7d-94a0-47491ed241d6",
    "stream_url": null,
    "component_id": "ChatOutput-aF5lw",
    "files": [],
    "type": "text"
  }
],
"component_display_name": "Chat Output",
"component_id": "ChatOutput-aF5lw",
"used_frozen_result": false
}
]
}
}

```

В производственном приложении вам, вероятно, потребуется извлекать нужные части ответа для возврата пользователю, сохранения в журналах и т.д.

Извлечение данных из ответа

Следующий пример основан на фрагменте из панели API и создаёт чат вопросов и ответов в терминале, сохраняя предыдущий ответ агента.

1. Включите фрагмент `/run` вашего потока **Simple Agent** в следующий скрипт:

Python:

```
import requests
```

```
url = "http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID"
```

```
def ask_agent(question):
    payload = {
        "output_type": "chat",
        "input_type": "chat",
        "input_value": question,
    }
    headers = {
        "Content-Type": "application/json",
        "x-api-key": "ASTRA_MULTIAGENT_AI_API_KEY"
    }
    try:
        response = requests.post(url, json=payload, headers=headers)
        response.raise_for_status()
        data = response.json()
        return data["outputs"]["outputs"]["message"]["message"]
    except Exception as e:
        return f"Error: {str(e)}"
```

```
previous_answer = None
```

```
while True:
    print("\nЗадайте агенту любой вопрос, например: 'What is 15 * 7?' или 'What is the capital of France?'"
    print("Введите 'quit' для выхода или 'compare' для сравнения с предыдущим ответом")
    user_question = input("Ваш вопрос: ")

    if user_question.lower() == 'quit':
        break
    elif user_question.lower() == 'compare':
        if previous_answer:
            print(f"\nПредыдущий ответ: {previous_answer}")
        else:
            print("\nПредыдущего ответа нет!")
        continue

    result = ask_agent(user_question)
    print(f"\nОтвет агента: {result}")
    previous_answer = result
```

JavaScript:

```
const readline = require('readline');

const rl = readline.createInterface({
    input: process.stdin,
    output: process.stdout
});

const url = 'http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID';

let previousAnswer = null;

async function askAgent(question) {
    const payload = {
        "output_type": "chat",
        "input_type": "chat",
        "input_value": question
    };
    const options = {
        method: 'POST',
        headers: {
```



```

        'Content-Type': 'application/json',
        'x-api-key': 'ASTRA_MULTIAGENT_AI_API_KEY'
    },
    body: JSON.stringify(payload)
  };
  try {
    const response = await fetch(url, options);
    const data = await response.json();
    return data.outputs.outputs.outputs.message.message;
  } catch (error) {
    return `Error: ${error.message}`;
  }
}

async function startChat() {
  console.log("\nЗадайте агенту любой вопрос, например: 'What is 15 * 7?' или 'What is the capital of France?'");
  console.log("Введите 'quit' для выхода или 'compare' для сравнения с предыдущим ответом");

  const askQuestion = () => {
    rl.question("\nВаш вопрос: ", async (userQuestion) => {
      if (userQuestion.toLowerCase() === 'quit') {
        rl.close();
        return;
      }
      if (userQuestion.toLowerCase() === 'compare') {
        console.log(previousAnswer
          ? "\nПредыдущий ответ: ${previousAnswer}"
          : "\nПредыдущего ответа нет!");
        askQuestion();
        return;
      }
      const result = await askAgent(userQuestion);
      console.log("\nОтвет агента: ${result}");
      previousAnswer = result;
      askQuestion();
    });
  };

  askQuestion();
}

startChat();

```

2. Чтобы просмотреть предыдущий ответ агента, введите compare. Чтобы закрыть чат, введите quit.

Применение tweaks для временного переопределения параметров потока

Вы можете включать tweaks в запросы, чтобы временно изменить параметры потока. Tweaks добавляются к телу запроса API и временно изменяют параметры компонентов только для одного запуска — они не изменяют базовую конфигурацию потока и не сохраняются между запусками.

Tweaks добавляются в payload конечной точки /run. Для упрощения форматирования определите их в панели **Input Schema** Генеративный ИИ перед копированием фрагмента кода.

1. В панели **API access** нажмите **Input Schema**, чтобы открыть соответствующую панель.
2. Выберите параметр, который хотите изменить. Включение параметров в **Input Schema** не меняет их постоянно — оно лишь добавляет их в примеры кода.

3. Например, чтобы изменить модель LLM агента с OpenAI на Anthropic и включить ключ API Anthropic в запрос, выберите компонент **Agent** в панели **Input Schema** и включите поле **Language Model**.

Генеративный ИИ обновляет объект tweaks в фрагментах кода на основе ваших входных параметров. Используйте обновлённые фрагменты в скрипте для запуска потока с вашими переопределениями:

```
payload = {  
  "output_type": "chat",  
  "input_type": "chat",  
  "input_value": "hello world!",  
  "tweaks": {  
    "Agent-ZOknz": {  
      "agent_llm": "Anthropic",  
      "api_key": "ANTHROPIC_API_KEY",  
      "model_name": "claude-opus-4-5-20251101"  
    }  
  }  
}
```

4. Учебные примеры

4.1 Создание чат-бота с векторным RAG

Это руководство демонстрирует, как использовать Генеративный ИИ для создания чат-бота, применяющего Retrieval Augmented Generation (RAG): данные встраиваются в виде векторов в векторную базу данных, после чего с ними можно вести диалог.

Требования

- Установите и запустите Генеративный ИИ
- Создайте API-ключ Генеративный ИИ
- Создайте API-ключ OpenAI
- Установите JavaScript-клиент Генеративный ИИ
- Знакомство с концепциями векторного поиска: векторные базы данных и RAG

Создание потока векторного RAG

1. В Генеративный ИИ нажмите **Новый поток**, затем выберите шаблон **Векторное хранилище RAG**.

О шаблоне «Векторное хранилище RAG»

Шаблон содержит два потока.

Поток загрузки данных заполняет векторное хранилище данными из файла. Эти данные используются для ответа на запросы, отправленные в **поток поискового модуля**.

Конкретно, **поток загрузки данных** загружает данные из локального файла, разбивает их на фрагменты, индексирует в векторной базе данных и вычисляет эмбединги для каждого фрагмента, которые сохраняются вместе с данными. Этот поток нужно запускать только при загрузке новых данных в векторную базу данных.

Поток поискового модуля получает сообщение из чата, генерирует его эмбединг и с помощью нескольких компонентов восстанавливает фрагменты в текст. Ответ генерируется путём сравнения нового эмбединга с сохранёнными для поиска похожих данных.

2. Добавьте ваш API-ключ **OpenAI** в оба компонента **OpenAI Эмбединги**.
3. Необязательно: замените оба компонента векторного хранилища **Astra DB** на **Chroma DB** или другое векторное хранилище по вашему выбору. В этом руководстве используется Chroma DB.

Поток загрузки данных должен включать компоненты **Чтение файла**, **Разделить текст**, **Модель эмбедингов**, векторное хранилище (например, **Chroma DB**) и **Вывод чата**.

Поток поискового модуля должен включать компоненты **Ввод в чат**, **Модель эмбедингов**, векторное хранилище, **Парсер**, **Запрос**, **Языковая модель** и **Вывод чата**.

Потоки готовы к использованию. Следуйте руководству дальше, чтобы загрузить данные в векторное хранилище и вызвать поток из приложения чат-бота.

Загрузка данных и генерация эмбедингов

Загрузить данные и сгенерировать эмбединги можно через визуальный редактор или через эндпоинт `/v2/files`.

Вариант с визуальным редактором проще, но рекомендуется только в том случае, когда пользователь, создавший поток, сам же загружает в него данные.

Когда данные загружают несколько пользователей или требуется программная загрузка, используйте вариант с API Генеративный ИИ.

1. В потоке RAG-чат-бота нажмите компонент **Чтение файла**, затем нажмите **Файл**.
2. Выберите локальный файл для загрузки и нажмите **Open**. Файл будет загружен на сервер Генеративный ИИ.
3. Чтобы загрузить данные в векторное хранилище, нажмите компонент хранилища векторов, затем нажмите **Запустить компонент** — это выполнит выбранный компонент и все предшествующие зависимости.

Для программной загрузки данных используйте эндпоинты `/v2/files/` и `/v1/run/$FLOW_ID`. Первый загружает файл на сервер Генеративный ИИ и возвращает путь к нему. Второй запускает **поток загрузки данных**, ссылаясь на этот путь, чтобы разбить данные на фрагменты, создать эмбединги и сохранить их в векторном хранилище.

Пример формирования запросов загрузки файлов на Python см. в руководстве Создание чат-бота с загрузкой файлов. Ниже приведён скрипт, демонстрирующий этот процесс.

// Node 18+ example using global fetch, FormData, and Blob

// 1. Prepare the form data with the file to upload

```
const fileBuffer = await fs.readFile('FILE_NAME');
const data = new FormData();
data.append('file', new Blob([fileBuffer]), 'FILE_NAME');
const headers = { 'x-api-key': 'ASTRA_API_KEY' };
```

// 2. Upload the file to Генеративный ИИ

```
const uploadRes = await fetch('ASTRA_SERVER_ADDRESS/api/v2/files/', {
  method: 'POST',
  headers,
  body: data
});
const uploadData = await uploadRes.json();
const uploadedPath = uploadData.path;
```

// 3. Call the run endpoint with the uploaded file path

```
const payload = {
  input_value: "Analyze this file",
  output_type: "chat",
  input_type: "text",
  tweaks: {
    'FILE_COMPONENT_NAME': {
      path: uploadedPath
    }
  }
};
const runRes = await fetch('ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json', 'x-api-key': 'ASTRA_API_KEY' },
  body: JSON.stringify(payload)
});
const astraData = await runRes.json();
// Output only the message
console.log(astraData.outputs?.?.outputs?.?.results?.message?.data?.text);
```

Когда поток запускается, он обрабатывает выбранный файл, разбивает данные на фрагменты, загружает их в векторное хранилище и генерирует эмбединги, которые также сохраняются в хранилище.

Векторная база данных теперь содержит данные с эмбедингами, которые LLM может использовать как контекст для ответа на запросы, как показано в следующем разделе.

Общение с потоком из JavaScript-приложения

Чтобы вести диалог с данными в векторном хранилище, создайте чат-бот, который программно запускает **поток поискового модуля**.

В этом руководстве для демонстрации используется JavaScript.

1. Соберите следующую информацию:

- ASTRA_SERVER_ADDRESS: домен вашего сервера Генеративный ИИ. Значение по умолчанию — 127.0.0.1:7860. Его можно найти в фрагментах кода на панели **Доступ к API** вашего потока.
- FLOW_ID: UUID вашего потока или пользовательское имя эндпоинта. Его можно найти в фрагментах кода на панели **Доступ к API** вашего потока.
- ASTRA_API_KEY: действительный API-ключ Генеративный ИИ.

2. Скопируйте следующий скрипт в файл JavaScript и замените заполнители собранными значениями:

```
const readline = require('readline');
const { AstraClient } = require('@datastax/astra-client');

// pragma: allowlist nextline secret
const API_KEY = 'ASTRA_API_KEY';
const SERVER = 'ASTRA_SERVER_ADDRESS';
const FLOW_ID = 'FLOW_ID';

const rl = readline.createInterface({ input: process.stdin, output: process.stdout });

// Initialize the Astra client
const client = new AstraClient({
  baseUrl: SERVER,
  apiKey: API_KEY
});

async function sendMessage(message) {
  try {
    const response = await client.flow(FLOW_ID).run(message, {
      session_id: 'user_1'
    });
    // Use the convenience method to get the chat output text
    return response.chatOutputText() || 'No response';
  } catch (error) {
    return `Error: ${error.message}`;
  }
}

function chat() {
  console.log('Генеративный ИИ RAG Chatbot (type "quit" to exit)\n');

  const ask = () => {
    rl.question('👤 You: ', async (input) => {
      if (['quit', 'exit', 'bye'].includes(input.trim().toLowerCase())) {
        console.log('Goodbye!');
        rl.close();
        return;
      }

      const response = await sendMessage(input.trim());
      console.log('🤖 Assistant: ${response}\n');
      ask();
    });
  };
}
```

```
ask();  
}
```

```
chat();
```

Скрипт создаёт Node-приложение, которое общается с содержимым вашего векторного хранилища, используя типы ввода и вывода `chat` для взаимодействия с потоком. Контекст беседы сохраняется между несколькими сообщениями. Если использовать типы ввода и вывода `text`, каждый запрос будет отдельной строкой текста.

TypeScript-клиент Генеративный ИИ предоставляет удобный метод `chatOutputText()`, упрощающий работу со сложной JSON-структурой ответа. Вместо ручного обхода `data.outputs[0].outputs[0].results.message.data.text` клиент автоматически извлекает текстовое сообщение и корректно обрабатывает потенциально неопределённые значения.

3. Сохраните и запустите скрипт, чтобы отправить запросы и протестировать поток.

Результат

Ниже приведён пример ответа потока, использованного в этом руководстве. В зависимости от поведения LLM и входных данных ваш ответ может отличаться.

4.2 Создание чат-бота, который может принимать файлы

Это руководство показывает, как построить чат-бот, который читает загруженные файлы и отвечает на вопросы по их содержимому — например, по протоколам встреч или заявкам на вакансию.

Например, можно загрузить контракт и спросить: «Какие положения о расторжении содержатся в этом соглашении?» — или загрузить резюме и спросить: «Есть ли у этого кандидата опыт в маркетинговой аналитике?»

Основная цель руководства — показать, как передавать файлы как входные данные в поток Генеративный ИИ, чтобы чат-бот мог использовать их содержимое в ответах.

Требования

- Установите и запустите Генеративный ИИ
- Создайте API-ключ Генеративный ИИ
- Создайте API-ключ OpenAI

В руководстве используется LLM от OpenAI. Если вы хотите использовать другого поставщика, вам понадобятся действительные учётные данные для него.

Создание потока, принимающего файлы

Чтобы обрабатывать файлы, поток должен содержать компонент **Чтение файла**, подключённый к компоненту, принимающему входные данные, — например, к компоненту **Шаблон промпта** или **Агент**.

Следующие шаги изменяют шаблон **Основной промпт** так, чтобы он принимал файлы в качестве входных данных:

1. В Генеративный ИИ нажмите **Новый поток**, затем выберите шаблон **Основной промпт**.
2. В компоненте **Языковая модель** введите ваш API-ключ OpenAI.
Если вы хотите использовать другого поставщика или модель, отредактируйте поля **Поставщик модели**, **Название модели** и **Ключ API** соответственно.
3. Чтобы проверить действительность API-ключа, нажмите **Playground** и задайте LLM любой вопрос. LLM должен ответить в соответствии со спецификациями в поле **Шаблон** компонента **Шаблон промпта**.

4. Закройте **Playground** и измените компонент **Шаблон промпта**, чтобы он принимал файлы в дополнение к вводу чата. Для этого отредактируйте поле **Шаблон** и замените запрос по умолчанию следующим текстом:

```
ChatInput:
{chat-input}
File:
{file}
```

Вы можете использовать любые строки в качестве имён переменных шаблона. Эти строки становятся именами полей (входных портов) в компоненте **Шаблон промпта**.

В этом руководстве переменные названы по компонентам, к которым они подключаются: **chat-input** для компонента **Ввод в чат** и **file** для компонента **Чтение файла**. 5. Добавьте компонент **Чтение файла** в поток и подключите порт **Необработанный контент** к входному порту **file** компонента **Шаблон промпта**. Чтобы соединить порты, щёлкните на одном порту и перетащите соединение к другому.

Файлы можно добавить непосредственно в компонент **Чтение файла**, чтобы предварительно загрузить входные данные до запуска потока, либо загружать их во время выполнения. Загрузка файлов во время выполнения описана в следующем разделе.

На этом этапе поток состоит из пяти компонентов. Компоненты **Ввод в чат** и **Чтение файла** подключены к входным портам компонента **Шаблон промпта**. Выходной порт компонента **Шаблон промпта** подключается к входному порту компонента **Языковая модель**. Выходной порт компонента **Языковая модель** подключается к компоненту **Вывод чата**, который возвращает финальный ответ пользователю.

Отправка запросов в поток из Python-приложения

Этот раздел демонстрирует, как отправлять файлы в поток из приложения.

Для этого приложение должно отправить POST-запрос к `/run` на сервер Генеративный ИИ с файлом и текстовым промптом. Результат содержит данные о выполнении потока и ответ LLM.

В этом примере используется локальный экземпляр Генеративный ИИ, а LLM просят оценить образец резюме. Если у вас нет резюме под рукой, скачайте `fake-resume.txt`.

Пример формирования запросов загрузки файлов на JavaScript см. в руководстве Создание чат-бота с векторным RAG. 1. Соберите следующую информацию:

- **ASTRA_SERVER_ADDRESS**: домен вашего сервера Генеративный ИИ. Значение по умолчанию — `127.0.0.1:7860`. Его можно найти в фрагментах кода на панели **Доступ к API** вашего потока.
- **FLOW_ID**: UUID вашего потока или пользовательское имя эндпоинта. Его можно найти в фрагментах кода на панели **Доступ к API** вашего потока.
- **FILE_COMPONENT_ID**: UUID компонента **Чтение файла** в вашем потоке, например `File-KZP68`. Чтобы найти ID компонента, откройте поток в модуле «Генеративный ИИ», щёлкните компонент **Чтение файла**, затем нажмите **Управление**. ID компонента находится в верхней части панели **Управление**.
- **CHAT_INPUT**: сообщение для отправки во вход чата потока, например `Оцените это резюме на предмет вакансии в моём отделе маркетинга`.
- **FILE_NAME** и **FILE_PATH**: имя и путь к локальному файлу, который вы хотите отправить в поток.
- **ASTRA_API_KEY**: действительный API-ключ Генеративный ИИ.

2. Скопируйте следующий скрипт в файл Python и замените заполнители собранными значениями:

```
# 1. Set the upload URL
url = "http://ASTRA_SERVER_ADDRESS/api/v2/files/"

# 2. Prepare the file and payload
payload = {}
files = [
('file', ('FILE_PATH', open('FILE_NAME', 'rb'), 'application/octet-stream'))
]
headers = {
'Accept': 'application/json',
'x-api-key': 'ASTRA_API_KEY'
}

# 3. Upload the file to Генеративный ИИ
response = requests.request("POST", url, headers=headers, data=payload, files=files)
print(response.text)

# 4. Get the uploaded file path from the response
uploaded_data = response.json()
uploaded_path = uploaded_data.get('path')

# 5. Call the run endpoint with the uploaded file path
run_url = "http://ASTRA_SERVER_ADDRESS/api/v1/run/FLOW_ID"
run_payload = {
"input_value": "CHAT_INPUT",
"output_type": "chat",
"input_type": "chat",
"tweaks": {
"FILE_COMPONENT_ID": {
"path": uploaded_path
}
}
}
run_headers = {
'Content-Type': 'application/json',
'Accept': 'application/json',
'x-api-key': 'ASTRA_API_KEY'
}
run_response = requests.post(run_url, headers=run_headers, data=json.dumps(run_payload))
astra_data = run_response.json()
# Output only the message
message = None
try:
message = astra_data['outputs']['outputs']['results']['message']['data']['text']
except (KeyError, IndexError, TypeError):
pass
print(message)
```

Скрипт содержит два запроса.

Первый запрос загружает файл, например fake-resume.txt, на сервер Генеративный ИИ через эндпоинт /v2/files. Запрос возвращает путь к файлу, который используется в последующих запросах, например 02791d46-812f-4988-ab1c-7c430214f8d5/fake-resume.txt.

Второй запрос отправляет сообщение в поток Генеративный ИИ через эндпоинт /v1/run/. Параметр tweaks включает путь к загруженному файлу как переменную uploaded_path и передаёт файл напрямую в компонент **Чтение файла**.

3. Сохраните и запустите скрипт, чтобы отправить запросы и протестировать поток.

Начальный вывод содержит JSON-ответ от эндпоинта загрузки файла, включая внутренний путь хранения файла в модуле «Генеративный ИИ». Затем LLM извлекает файл и оценивает его содержимое — в данном случае пригодность резюме для вакансии.

4.3 Подключение приложений к агентам

Это руководство показывает, как подключить JavaScript-приложение к агенту Генеративный ИИ.

С помощью агента ваше приложение может использовать любые подключённые инструменты для получения более контекстных и актуальных данных без изменения кода приложения. Инструменты выбираются внутренней LLM (Large Language Model) агента для решения задач и ответов на вопросы.

Требования

- Установите и запустите Генеративный ИИ
- Создайте ключ API Генеративный ИИ
- Установите JavaScript-клиент Генеративный ИИ
- Создайте ключ API OpenAI

В этом руководстве используется LLM от OpenAI. Если вы хотите использовать другого поставщика, вам понадобятся действительные учётные данные для этого поставщика.

Создание потока агента

Следующие шаги изменяют шаблон **Простого Агента**, чтобы подключить компонент **Папка** и компонент **Веб-поиск** в качестве инструментов для компонента **Агент**. Компонент **Папка** загружает все файлы заданного типа из целевого каталога на вашем локальном компьютере, а компонент **Веб-поиск** выполняет поиск в DuckDuckGo. При подключении к компоненту **Агент** в качестве инструментов агент может использовать эти компоненты при обработке запросов.

1. В Генеративный ИИ нажмите **Новый Поток**, затем выберите шаблон **Простой Агент**.
2. Удалите инструменты **URL** и **Калькулятор**, затем добавьте компоненты **Папка** и **Веб-поиск** в ваш поток.
3. В поле **Путь** компонента **Папка** укажите путь к каталогу и типы файлов, которые вы хотите сделать доступными для компонента **Агент**.

В этом руководстве агенту нужен доступ к записи покупок клиентов, поэтому имя каталога — `customer_orders`, а тип файлов — `.csv`. Позже в руководстве агенту будет предложено найти значения email в данных о клиентах.

Вы можете адаптировать руководство под свои данные, либо, чтобы следовать инструкциям, загрузите `customer_orders.csv`

4. В меню заголовков компонентов **Папка** и **Веб-поиск** включите **Режим Инструмента**, чтобы можно было использовать компоненты с агентом.
5. Подключите порты **Набора инструментов** компонентов **Папка** и **Веб-поиск** к порту **Инструменты** компонента **Агент**.
6. В компоненте **Агент** введите ваш ключ API OpenAI.

Если вы хотите использовать другого поставщика или модель, отредактируйте соответствующим образом поля **Поставщик модели**, **Название модели** и **Ключ API**.

7. Чтобы протестировать поток, нажмите **Playground**, затем задайте LLM вопрос, например `Порекомендуйте 3 товара от carol.davis@example.com, основываясь на предыдущих заказах.`

Для данного примера запроса LLM ответит рекомендациями и веб-ссылками на товары, основанными на предыдущих заказах в `customer_orders.csv`.

Playground выводит цепочку рассуждений агента, когда он выбирает инструменты и взаимодействует с их функциональностью. Например, агент может использовать инструмент `as_dataframe` компонента **Папка** для получения `DataFrame`, а инструмент `perform_search` компонента **Веб-поиск** — для поиска ссылок на связанные товары.

Добавить компонент Шаблон промпта в поток

В этом примере приложение отправляет адрес электронной почты клиента агенту Генеративный ИИ. Агент сравнивает предыдущие заказы клиента в компоненте **Папка**, ищет в интернете использованные версии этих товаров и возвращает три результата.

1. Чтобы включить адрес электронной почты в качестве значения в ваш поток, добавьте компонент **Шаблон промпта** в ваш поток между компонентами **Ввод в чат** и **Агент**.
2. В поле **Шаблон** компонента **Шаблон промпта** введите: `Порекомендуй 3 бывших в употреблении товара для {email}, основываясь на предыдущих заказах. Добавление значения {email} в фигурных скобках создаёт новый ввод в компоненте Шаблон промпта`, а компонент, подключённый к порту `{email}`, поставляет значение для этой переменной. Это создаёт точку входа адреса электронной почты пользователя в поток из вашего запроса. Если вы не используете примерный файл `customer_orders.csv`, измените ввод, чтобы искать значение в вашем наборе данных.

На данном этапе ваш поток состоит из шести компонентов. Компонент **Ввод в чат** подключён к входному порту `email` компонента **Шаблон промпта**. Затем выход компонента **Шаблон промпта** подключён к входному порту **Системное сообщение** компонента **Агент**. Компоненты **Папка** и **Веб-поиск** подключены к порту **Инструменты** компонента **Агент**. Наконец, выход компонента **Агент** подключён к компоненту **Вывод чата**, который возвращает окончательный ответ приложению.

Отправка запросов в ваш поток из JavaScript-приложения

После того как ваш поток работает, подключите его к JavaScript-приложению, чтобы использовать ответы агента.

1. Чтобы создать JavaScript-приложение для подключения к вашему потоку, соберите следующую информацию:
 - `ASTRA_SERVER_URL_ADDRESS`: Домен вашего сервера Генеративный ИИ. Значение по умолчанию — `127.0.0.1:7860`. Вы можете получить это значение из фрагментов кода на вкладке **API access** вашего потока.
 - `FLOW_ID`: `UUID` вашего потока или пользовательское имя конечной точки. Вы можете получить это значение из фрагментов кода на вкладке **API access** вашего потока.
 - `ASTRA_API_KEY`: Действительный ключ API Генеративный ИИ.
2. Скопируйте следующий скрипт в файл JavaScript, а затем замените заполнители собранной на предыдущем шаге информацией. Если вы используете примерный файл `customer_orders.csv`, вы можете запустить этот пример как есть с примерным адресом электронной почты в образце кода. Если нет, измените `const email = "isabella.rodriguez@example.com"`, чтобы искать значение в вашем наборе данных.

```
import { AstraClient } from "@datastax/Генеративный ИИ-client";
```

```
const ASTRA_SERVER_URL_ADDRESS = 'ASTRA_SERVER_URL_ADDRESS';
const FLOW_ID = 'FLOW_ID';
const ASTRA_API_KEY = 'ASTRA_API_KEY';
```

```

const email = "isabella.rodriguez@example.com";

async function runAgentFlow(): Promise {
  try {
    // Initialize the Генеративный ИИ client
    const client = new AstraClient({
      baseUrl: ASTRA_SERVER_URL_ADDRESS,
      apiKey: ASTRA_API_KEY
    });

    console.log(`Connecting to Генеративный ИИ server at: ${ASTRA_SERVER_URL_ADDRESS} `);
    console.log(`Flow ID: ${FLOW_ID}`);
    console.log(`Email: ${email}`);

    // Get the flow instance
    const flow = client.flow(FLOW_ID);

    // Run the flow with the email as input
    console.log(`\nSending request to agent...`);
    const response = await flow.run(email, {
      session_id: email // Use email as session ID for context
    });

    console.log(`\n=== Response from Генеративный ИИ ===`);
    console.log(`Session ID:`, response.sessionId);

    // Extract URLs from the chat message
    const chatMessage = response.chatOutputText();
    console.log(`\n=== URLs from Chat Message ===`);
    const messageUrls = chatMessage.match(/https?:\/\/[^\s"]+/g) || [];
    const cleanMessageUrls = [...new Set(messageUrls)].map(url => url.trim());
    console.log(`URLs from message:`);
    cleanMessageUrls.slice(0, 3).forEach(url => console.log(url));

  } catch (error) {
    console.error(`Error running flow:`, error);

    // Provide error messages
    if (error instanceof Error) {
      if (error.message.includes('fetch')) {
        console.error(`\nMake sure your Генеративный ИИ server is running and accessible at:`,
ASTRA_SERVER_URL_ADDRESS);
      }
      if (error.message.includes('401') || error.message.includes('403')) {
        console.error(`\nCheck your API key configuration`);
      }
      if (error.message.includes('404')) {
        console.error(`\nCheck your Flow ID - make sure it exists and is correct`);
      }
    }
  }
}

// Run the function
console.log(`Starting Генеративный ИИ Agent...\n`);
runAgentFlow().catch(console.error);

```

3. Сохраните и запустите скрипт, чтобы отправить запрос и протестировать поток.

Ваше приложение получает три URL-адреса с рекомендациями по использованным товарам на основе предыдущих заказов клиента в вашем локальном CSV, без изменения кода.

Результат

Ниже приведён пример ответа из потока этого руководства. Из-за особенностей LLM и вариаций ваших входных данных ваш ответ может отличаться.

Starting Генеративный ИИ Agent...

Connecting to Генеративный ИИ server at: http://localhost:7860

Flow ID: local-db-search

Email: isabella.rodriguez@example.com

Sending request to agent...

=== Response from Генеративный ИИ ===

Session ID: isabella.rodriguez@example.com

URLs found:

<https://www.facebook.com/marketplace/258164108225783/electronics/>

<https://www.facebook.com/marketplace/458332108944152/furniture/>

<https://www.facebook.com/marketplace/613732137493719/kitchen-cabinets/>

4. Чтобы быстро проверить трафик к вашему потоку, откройте **Playground**. Новые сессии именуются по адресу электронной почты пользователя. Сохранение уникальности сессий помогает агенту поддерживать контекст. Подробнее об идентификаторах сессий см. в Идентификатор сеанса.

4.4 Подключение к серверам MCP из приложения

Это руководство показывает, как подключать серверы MCP к вашим приложениям с помощью компонента **Инструменты MCP** в модуле «Генеративный ИИ».

Протокол контекста модели (MCP) помогает агентам интегрироваться с языковыми моделями через *клиенты MCP* и *серверы MCP*. Серверы MCP предоставляют инструменты, которые агенты (клиенты MCP) используют для выполнения специализированных задач. Серверы MCP подключаются к клиентам MCP, таким как Cursor, — вы взаимодействуете с клиентом, а клиент при необходимости использует инструменты подключённых серверов для обработки ваших запросов.

Генеративный ИИ можно запускать как в роли клиента MCP, так и в роли сервера MCP:

- Генеративный ИИ в роли клиента MCP — компонент **Агент** в потоке может использовать подключённые компоненты в качестве инструментов для обработки запросов, включая инструменты любого подключённого сервера MCP.
- Генеративный ИИ в роли сервера MCP — ваши потоки становятся инструментами, которые может использовать клиент MCP: внешний или другой поток Генеративный ИИ.

В этом руководстве вы подключите несколько серверов MCP к потоку с помощью компонента **Инструменты MCP**, а затем используете Python-приложение для запуска потока и программного взаимодействия с агентом.

Требования

- Установите и запустите Генеративный ИИ
- Создайте ключ API Генеративный ИИ
- Создайте ключ API OpenAI

В этом руководстве используется языковая модель OpenAI. Если вы хотите использовать другого провайдера, подготовьте соответствующие учётные данные.

Создание потока агента

1. В Генеративный ИИ нажмите **Новый поток** и выберите шаблон **Простой агент**.

2. В компоненте **Агент** введите ваш ключ API OpenAI.

Если вы хотите использовать другого провайдера или модель, измените поля **Провайдер модели**, **Название модели** и **Ключ API** соответственно.

3. Чтобы протестировать поток, нажмите **Playground** и задайте языковой модели вопрос: Is it safe to go hiking in the Adirondacks today?

Этот запрос показывает, что языковая модель сама по себе может не иметь доступа к информации для обработки специализированных запросов. В данном случае модель OpenAI даёт расплывчатый ответ, хотя агент знает текущую дату благодаря внутренней функции `get_current_date`.

Чтобы улучшить ответ, можно подключить к потоку серверы MCP, предоставляющие специализированные инструменты. В следующей части руководства вы подключите сервер MCP с данными о погоде в реальном времени, чтобы агент мог формировать более конкретные ответы.

Добавление компонента «Инструменты MCP»

Существует множество серверов MCP, предлагающих различные инструменты для разных задач. Чтобы использовать сервер MCP с клиентом MCP, необходимо сделать его доступным для клиента. Это можно сделать несколькими способами:

- Установить сервер локально.
- Использовать `uvx` или `prx` для скачивания и запуска пакета сервера.
- Обратиться к серверу, работающему удалённо, например к тем, что доступны в Smithery.

В этом руководстве показано, как установить сервер погоды локально с помощью `uv` `pip` `install` и как запустить пакет сервера геолокации через `prx`. Требования конкретного сервера MCP могут отличаться.

В Генеративный ИИ для подключения конкретного сервера MCP к потоку используется компонент **Инструменты MCP**. Для каждого сервера MCP, который вы хотите использовать в потоке, требуется отдельный компонент **Инструменты MCP**.

1. Для этого руководства установите `weather MCP server` на локальном компьютере с помощью `uv` и Python:

```
uv pip install mcp_weather_server
```

Убедитесь, что сервер установлен в той же среде Python, где запущен Генеративный ИИ:

- **Виртуальное окружение** — активируйте окружение перед установкой сервера.
- **Docker-образ Генеративный ИИ** — установите сервер внутри контейнера Docker.
- **Генеративный ИИ Desktop** или **системный Генеративный ИИ OSS** — установите сервер глобально или в той же пользовательской среде, где запускается Генеративный ИИ.

2. В потоке **Простой агент** удалите инструменты **URL** и **Калькулятор**, затем добавьте компонент **Инструменты MCP**.

3. Нажмите на компонент **Инструменты MCP**, затем нажмите: **Добавить сервер MCP**.

Чтобы управлять всеми подключениями серверов MCP, нажмите **Серверы MCP** в визуальном редакторе или откройте значок профиля → **Настройки** → **Серверы MCP**. 4. В панели: **Добавить сервер MCP** укажите команду запуска и аргументы для подключения сервера погоды. Используйте вариант **JSON** или **STDIO**.

Генеративный ИИ выполняет команду запуска сервера, когда агент решает использовать инструмент этого сервера. Обе конфигурации передают одинаковую информацию в разных

форматах: если репозиторий вашего сервера MCP предоставляет только JSON-файл, вы всё равно можете использовать эти значения с опцией STDIO.

Чтобы предоставить конфигурацию сервера MCP в виде JSON-объекта, выберите **JSON** и вставьте конфигурацию в поле **JSON**:

```
{
  "mcpServers": {
    "weather": {
      "command": "python",
      "args": [
        "-m",
        "mcp_weather_server"
      ],
      "disabled": false,
      "autoApprove": []
    }
  }
}
```

Чтобы предоставить конфигурацию сервера MCP в графическом формате, выберите **STDIO** и введите значения в указанные поля:

- **Название:** weather
 - **Команда:** python
 - **Аргументы:**
 - -m
 - mcp_weather_server
5. Нажмите: **Добавить сервер** и дождитесь заполнения списка **Действия** — это означает, что сервер MCP успешно подключён.

С этим сервером погоды компонент **Инструменты MCP** также добавляет необязательное поле **Город**. Для данного руководства оставьте его пустым: на следующем шаге вы добавите сервер геолокации, который агент будет использовать для определения местоположения.

6. Нажмите на компонент **Инструменты MCP**, включите **Режим инструмента** в меню заголовка компонента, затем подключите порт **Набор инструментов** компонента к порту **Инструменты** компонента **Агент**.

На данном этапе поток содержит четыре подключённых компонента:

- Компонент **Ввод в чат** подключён к порту **Ввод** компонента **Агент** — это позволяет запускать поток при получении входного запроса.
 - Компонент **Инструменты MCP** с сервером погоды подключён к порту **Инструменты** компонента **Агент** — агент использует это соединение только если решит, что сервер поможет ответить на запрос.
 - Порт **Вывод** компонента **Агент** подключён к компоненту **Вывод чата**, который возвращает окончательный ответ.
7. Чтобы протестировать сервер погоды, нажмите **Playground** и задайте вопрос: Is it safe to go hiking in the Adirondacks today?

В **Playground** отображается логика агента при анализе запроса и выборе инструментов. Благодаря дополнительному контексту от сервера погоды ответ становится более конкретным.

Серверы MCP часто настраиваются под конкретные задачи, такие как высокоспециализированные действия или цепочки инструментов для решения сложных многопроцессных задач. Обычно вам пришлось бы написать пользовательский скрипт для конкретной задачи, возможно включающий несколько вызовов API в одном скрипте, а затем

вам пришлось бы либо выполнить этот скрипт вне контекста агента, либо каким-то образом предоставить его агенту.

Вместо этого MCP гарантирует, что все серверы MCP добавляются к агентам одинаковым способом, без необходимости знать специфические структуры конечных точек каждого сервера или писать пользовательские интеграции. MCP — это стандартизированный способ интеграции множества разнообразных инструментов в агентные приложения. Вам не нужно изучать новый API или писать пользовательский код каждый раз, когда вы хотите использовать новый сервер MCP.

Кроме того, вы можете подключить множество серверов MCP к одному агенту, в зависимости от задач, которые вы хотите решить в вашем приложении. Чем больше серверов вы добавляете, тем более специализированный контекст агент может использовать в своих ответах. В этом руководстве добавление сервера погоды MCP уже улучшило качество ответа LLM.

В следующем разделе руководства вы добавите сервер `ip_geolocation` MCP, чтобы агент мог определить местоположение пользователя, если оно не указано в запросе.

Добавить сервер геолокации

Сервер Toolkit MCP `server` включает несколько инструментов MCP для мониторинга сети, в том числе IP-геолокацию. Он не обладает высокой точностью, но не требует API-ключа.

Обратите внимание, что этот инструмент возвращает IP-геолокацию вашего сервера Генеративный ИИ, поэтому если ваш сервер развернут удалённо, рассмотрите альтернативные подходы для получения данных о местоположении конкретного пользователя, такие как API геолокации браузера.

Этот сервер MCP можно запустить одной командой `prx`, которая скачивает и запускает пакет Toolkit MCP `server Node registry package` без локальной установки.

Чтобы добавить сервер Toolkit MCP в ваш поток, выполните следующее:

1. Добавьте ещё один компонент **Инструменты MCP** в ваш поток, щёлкните по компоненту, а затем нажмите: **Добавить Сервер MCP**.
2. Выберите **STDIO**.
3. Для **Название** введите `ip_geolocation`.

Название инструмента и его описание помогают агенту выбирать инструменты. Если ваш агент испытывает трудности с выбором инструментов, убедитесь, что названия и описания ясны и читаемы человеком.

4. Для **Команда** введите `prx @cyanheads/toolkit-mcp-server`.
4. Нажмите: **Добавить Сервер**, затем дождитесь заполнения списка **Действия**. Это означает, что сервер MCP успешно подключён.
5. Щёлкните компонент **Инструменты MCP**, включите **Инструмент Режим** в меню заголовка компонента, а затем подключите Порт компонента **Набор инструментов** к порту **Инструменты** компонента **Агент**.

Теперь ваш поток содержит дополнительный компонент **Инструменты MCP**, всего их пять.

Создайте Python-приложение, которое подключается к модулю «Генеративный ИИ»

На данном этапе вы можете открыть **Playground** и спросить о погоде в вашем текущем месте, чтобы протестировать инструмент IP-геолокации. Однако инструменты геолокации наиболее полезны в приложениях, где вы или ваши пользователи хотят узнавать погоду в разных точках мира.

В последней части этого руководства вы узнаете, как использовать API Генеративный ИИ для запуска потока в скрипте. Это может быть частью более крупного приложения, например мобильного, где пользователи хотят узнать, подходит ли погода для определённого вида спорта.

При использовании API Генеративный ИИ для запуска потока вы можете изменять некоторые аспекты потока без изменения кода. Например, вы можете добавить больше серверов MCP в ваш поток в модуле «Генеративный ИИ», а затем использовать тот же скрипт для его запуска. Вы можете использовать тот же ввод или новый ввод, который подсказывает агенту использовать другие инструменты.

1. Для Python-скрипта в этом руководстве соберите следующую информацию:
 - `ASTRA_SERVER_URL_ADDRESS`: Домен вашего сервера Генеративный ИИ. Значение по умолчанию — `127.0.0.1:7860`. Вы можете получить это значение из фрагментов кода на панели **API access** вашего потока.
 - `FLOW_ID`: UUID вашего потока или пользовательское имя конечной точки. Вы можете получить это значение из фрагментов кода на панели **API access** вашего потока.
 - `ASTRA_API_KEY`: Действительный ключ API Генеративный ИИ.
2. Скопируйте следующий скрипт в файл Python, а затем замените заполнители собранной на предыдущем шаге информацией:

```
import requests
import os

url = "ASTRA_SERVER_URL_ADDRESS/api/v1/run/FLOW_ID" # The complete API endpoint URL for this flow

# Request payload configuration
payload = {
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "What's the weather like where I am right now?"
}

# Request headers
headers = {
    "Content-Type": "application/json",
    "x-api-key": "ASTRA_API_KEY"
}

try:
    # Send API request
    response = requests.request("POST", url, json=payload, headers=headers)
    response.raise_for_status() # Raise exception for bad status codes

    # Parse and print only the message text
    data = response.json()
    message = data["outputs"][0]["outputs"][0]["results"]["message"]["text"]
    print(message)

except requests.exceptions.RequestException as e:
    print(f'Error making API request: {e}')
except ValueError as e:
    print(f'Error parsing response: {e}')
except (KeyError, IndexError) as e:
    print(f'Error extracting message from response: {e}')
```


Обратите внимание, что этот скрипт использует другой запрос, чем примеры в предыдущем **Playground**. В этом скрипте `input_value` запрашивает погоду в текущем местоположении пользователя, не предоставляя никаких подсказок о местоположении пользователя, например конкретного города.

Кроме того, в этом скрипте присутствует код парсинга для извлечения ответа LLM из полного ответа API Генеративный ИИ. Вам понадобится использовать аналогичное извлечение в своих приложениях, поскольку ответ API Генеративный ИИ содержит метаданные и другую информацию, не относящуюся к ответу, передаваемому пользователю.

3. Сохраните и запустите скрипт, чтобы отправить запрос и протестировать поток.

Агент использует инструмент `ip_geolocation` для определения местоположения запрашивающего, а затем использует инструмент `weather` для получения информации о погоде в этом месте. Например:

The weather in Waynesboro, Pennsylvania, is currently overcast with a temperature of 23.0°C (about 73.4°F).
If you need more details or have any other questions, feel free to ask!

Помните, что инструмент `ip_geolocation`, используемый в этом руководстве, использует местоположение вашего сервера Генеративный ИИ, которое может отличаться от вашего реального местоположения.

5. Визуальный редактор

Визуальный редактор Генеративный ИИ предназначен для создания, тестирования и совместного использования потоков — функциональных представлений рабочих процессов приложений. Потоки состоят из компонентов, представляющих отдельные шаги рабочего процесса.

Интерфейс перетаскивания позволяет создавать сложные AI-рабочие процессы без написания обширного кода. Вы можете подключать различные ресурсы: промпты, большие языковые модели (LLM), источники данных, агентов, MCP-серверы и другие инструменты и интеграции.

Чтобы создать и запустить первый поток за несколько минут, см. Быстрый старт Генеративный ИИ.

5.1 Рабочее пространство

При построении потока вы в основном взаимодействуете с рабочим пространством. Здесь вы добавляете компоненты, настраиваете их и соединяете между собой.

Из рабочего пространства также доступны **Playground**, меню **Поделиться** и **Логи**.

Жесты и взаимодействия

Сочетания клавиш, жесты и функции для навигации по рабочему пространству:

- **Горизонтальное и вертикальное панорамирование:** щёлкните и перетащите пустую область рабочего пространства.
- **Перестановка компонентов:** щёлкните и перетащите компонент в любое место рабочего пространства.

Чтобы изменить программную связь между компонентами, управляйте *рёбрами* или *портами* компонента. Подробнее см. в разделе Обзор компонентов.

Чтобы включить направляющие линии, нажмите **Помощь**, затем включите **Помощник**.

Если компоненты недоступны для редактирования, убедитесь, что поток разблокирован.

- **Масштабирование:** прокручивайте колесо мыши или трекпад либо нажмите **Элементы управления** рядом с процентом масштабирования для доступа к дополнительным параметрам: **Приблизить**, **Отдалить**, **100%** и **По размеру**.
- **Заметки и комментарии:** нажмите **Добавить заметку**.
- **Горячие клавиши:** нажмите **Помощь**, затем выберите **Сочетания клавиш**.

5.2 Playground

Если поток содержит компонент **Ввод в чат**, вы можете использовать **Playground** для запуска потока, ведения чата, просмотра входных и выходных данных, а также изменения памяти LLM для настройки ответов в реальном времени.

Чтобы попробовать это самостоятельно, создайте поток на основе шаблона **Простой агент** и нажмите **Playground** прямо в рабочем пространстве.

Если поток содержит компонент **Агент**, **Playground** отображает вызовы инструментов и их результаты — это помогает отслеживать использование инструментов и понимать логику ответов агента.

Подробнее см. в разделе Тестирование потоков в Playground.

5.3 Поделиться

Меню **Поделиться** предоставляет следующие варианты интеграции потока во внешние приложения:

- **API-доступ:** интегрируйте поток в приложения с помощью автоматически сгенерированных фрагментов кода на Python, JavaScript и curl.
- **Экспорт:** экспортируйте поток на локальный компьютер в виде JSON-файла.
- **МСП-сервер:** откройте поток как инструмент для МСП-совместимых клиентов.
- **Встроить на сайт:** встройте поток в приложения на HTML, React или Angular.
- **Публичный Playground:** поделитесь интерфейсом **Playground** с другим пользователем.

Публичный Playground предназначен исключительно для совместного тестирования, а не для запуска потока в производственном приложении.

Публичный Playground недоступен в модуле «Генеративный ИИ» Desktop.

6. Создание потоков

Поток — это функциональное представление рабочего процесса приложения. Потоки получают ввод, обрабатывают его и генерируют вывод.

Потоки состоят из *компонентов*, представляющих отдельные шаги рабочего процесса.

Потоки Генеративный ИИ полностью сериализуемы: их можно сохранять и загружать из файловой системы, где установлен Генеративный ИИ.

Чтобы создать и запустить первый поток за несколько минут, см. Быстрый старт Генеративный ИИ.

6.1 Создание потока

Со страницы **Проекты** доступны четыре способа создать поток:

- **Создать пустой поток:** выберите проект, нажмите **Новый поток**, затем **Пустой поток**.
- **Создать поток из шаблона:** выберите проект, нажмите **Новый поток**, затем выберите нужный шаблон.

Что такое шаблоны?

Шаблоны — это готовые потоки, которые можно использовать как отправную точку для собственного потока. Они варьируются от простых потоков с несколькими компонентами до сложных потоков с множеством компонентов и подпотоков.

Например, шаблон **Основной промпт** демонстрирует небольшой поток, который передаёт в LLM и ввод чата, и заранее определённые инструкции (промпт). Шаблон **Векторное хранилище RAG** состоит из двух подпотоков, демонстрирующих создание чат-бота с Retrieval Augmented Generation (RAG): один подпоток заполняет векторное хранилище релевантными данными и эмбедингами, второй запрашивает его для получения похожих данных в ответ на вопросы пользователя.

Вы также можете добавлять собственные шаблоны в кодовую базу Генеративный ИИ.

- **Дублировать существующий поток:** найдите нужный поток, нажмите **Ещё**, затем выберите **Дублировать**.
- **Импортировать поток:** см. Импорт и экспорт потоков.

Вы также можете создать поток с помощью API Генеративный ИИ, однако команда Генеративный ИИ рекомендует начинать с визуального редактора, пока вы не освоите процесс создания потоков.

Добавление компонентов

Потоки состоят из компонентов — узлов, которые вы настраиваете и соединяете в рабочем пространстве. Каждый компонент выполняет конкретную задачу: обращается к AI-модели, подключает источник данных и т. д.

Перетаскивайте компоненты из меню **Основные компоненты** и **Пакеты**, чтобы добавить их в поток. Затем настройте параметры компонентов и соедините их между собой.

Каждый компонент имеет параметры конфигурации: часть из них общая для всех компонентов, часть — уникальная для конкретного компонента.

Чтобы сформировать связный поток, соединяйте компоненты через *рёбра* или *порты*, у каждого из которых есть определённый тип принимаемых или отправляемых данных. Например, порты сообщений передают текстовые строки между компонентами.

Подробнее о конфигурации компонентов, типах портов и исходном коде компонентов см. в разделе Обзор компонентов.

Запуск потока

После создания прототипа потока протестируйте его в **Playground**. Когда будете готовы использовать Генеративный ИИ в разработке приложений, изучите запуск потоков через API Генеративный ИИ, более продвинутые параметры конфигурации — например, пользовательские зависимости, — и в конечном итоге контейнеризацию приложения Генеративный ИИ.

Для перехода в продакшн или развёртывания MCP-сервера Генеративный ИИ с публичным доступом см. Обзор развёртывания Генеративный ИИ.

Графы потока

При запуске потока Генеративный ИИ создаёт объект ориентированного ациклического графа (DAG) из узлов (компонентов) и рёбер (соединений), а узлы сортируются для определения порядка выполнения.

При построении графа для каждого компонента вызывается функция `def_build` — для валидации и подготовки узлов. Затем граф обрабатывается в порядке зависимостей: каждый узел создаётся и выполняется последовательно, а результаты передаются зависящим от них узлам.

6.2 Управление потоками в проектах

Страница **Проекты** — стартовая страница при запуске Генеративный ИИ. Отсюда вы можете управлять потоками и MCP-серверами.

Проекты Генеративный ИИ работают как папки для организации связанных потоков. Проект по умолчанию — **Стартовый проект**; новые потоки сохраняются в нём, если вы не создадите другой. Чтобы создать проект, нажмите **Создать новый проект**.

Чтобы вернуться на страницу **Проекты** после редактирования потока, нажмите название проекта или значок Генеративный ИИ в заголовке.

Редактирование параметров потока

1. На странице **Проекты** найдите нужный поток.
2. Нажмите **Ещё**, затем выберите **Редактировать параметры**.
3. Отредактируйте **Название** и **Описание**, затем нажмите **Сохранить**.

Блокировка потока

Чтобы предотвратить изменения потока, заблокируйте его:

1. На странице **Проекты** найдите нужный поток.
2. Нажмите **Ещё**, затем выберите **Редактировать параметры**.
3. Включите **Заблокировать поток**, затем нажмите **Сохранить**.

Повторите эти шаги, чтобы разблокировать поток, отключив **Заблокировать поток**.

При редактировании потока поле **Статус блокировки** отображает текущее состояние: **Заблокирован** или **Разблокирован**. Изменить статус блокировки непосредственно во время редактирования нельзя.

Перемещение потока

Чтобы переместить поток из одного проекта в другой:

1. На странице **Проекты** найдите нужный поток.
2. Перетащите его из списка потоков к названию целевого проекта в списке проектов.

Удаление потока

1. На странице **Проекты** найдите нужный поток.
2. Нажмите **Ещё**, затем выберите **Удалить**.

6.3 Хранение потоков и журналы

По умолчанию потоки и данные их выполнения хранятся в базе данных Генеративный ИИ, а журналы сохраняются вместе с остальными журналами Генеративный ИИ в каталоге конфигурации. Подробнее см. в разделах Управление памятью и Логирование.

7. Запуск потоков

7.1 Запуск потоков через API Генеративный ИИ

После того как вы создадите поток, вероятно, захотите интегрировать его в приложение — например, в чат-бот мобильного приложения или сайта.

- Запуск потоков через API Генеративный ИИ
- Встраивание виджета чата на сайт
- Обслуживание потоков через MCP-сервер Генеративный ИИ
- Запуск потоков через совместимый с OpenAI Responses endpoint

Хотя эти варианты можно использовать с изолированным локальным экземпляром Генеративный ИИ, они наиболее полезны, когда вы развернули сервер Генеративный ИИ или упаковали Генеративный ИИ как зависимость приложения.

API Генеративный ИИ

API Генеративный ИИ — основной способ программного доступа к потокам и серверам Генеративный ИИ.

Пример скрипта, вызывающего API Генеративный ИИ, см. в Быстром старте.

Генерация фрагментов кода для API

Для упрощения интеграции Генеративный ИИ автоматически генерирует фрагменты кода на Python, JavaScript и curl для каждого потока. Чтобы получить эти фрагменты, выполните следующее:

1. В Генеративный ИИ откройте поток, который хотите интегрировать в приложение.
2. Нажмите **Поделиться**, затем выберите **API-доступ**.

Фрагменты вызывают эндпоинт `/v1/run/$FLOW_ID` и автоматически подставляют минимально необходимые значения: URL сервера Генеративный ИИ, ID потока, заголовки и параметры запроса.

Пути в панели **API-доступ** предполагают среду `*nix`. Если вы используете Microsoft Windows или WSL, возможно, потребуется скорректировать пути к файлам во фрагментах кода.

3. Опционально: нажмите **Схема ввода**, чтобы изменить параметры компонентов во фрагментах кода, не затрагивая сам поток.
3. Скопируйте фрагмент для нужного языка.
4. Запустите фрагмент как есть или используйте его в контексте более крупного скрипта.

Дополнительные примеры других эндпоинтов API см. в разделе Начало работы с API Генеративный ИИ.

Аутентификация API Генеративный ИИ

В Генеративный ИИ версии 1.5 и новее большинство эндпоинтов API требуют аутентификации с помощью API-ключа Генеративный ИИ.

Фрагменты кода, сгенерированные в панели **API-доступ**, включают скрипт, проверяющий наличие переменной окружения `ASTRA_API_KEY` в текущем сеансе терминала. Этот скрипт не проверяет API-ключи, заданные вне текущего сеанса.

Чтобы скрипт работал, задайте переменную `ASTRA_API_KEY` в сеансе терминала, где планируете запускать фрагмент: например, `export ASTRA_API_KEY="sk..."`.

В качестве альтернативы отредактируйте фрагмент, добавив заголовок `x-api-key` для аутентификации запроса.

Для получения дополнительной информации см. API-ключи и аутентификация и Начало работы с API Генеративный ИИ.

Схема ввода (настройки)

Настройки — это разовые переопределения, изменяющие параметры компонентов во время выполнения без постоянного изменения потока. Пример использования настроек в скрипте см. в Быстром старте.

Настройки делают потоки более динамичными и переиспользуемыми.

Вы можете создать один поток и использовать его в нескольких приложениях, передавая специфичные для каждого приложения настройки в запросах к API Генеративный ИИ. В панели **API-доступ** нажмите **Схема ввода**, чтобы добавить tweaks в тело запроса во фрагментах кода потока.

Изменения в **Схеме ввода** сохраняются исключительно как настройки для фрагментов кода **API-доступа** данного потока. Они не изменяют параметры потока, заданные в рабочем пространстве, и не применяются к другим потокам.

Использование панели **Схема ввода** помогает избежать ошибок форматирования настроек, которые вы вручную добавляете в запросы к API Генеративный ИИ.

Например, следующая команда curl включает настройку, отключающую параметр **Сохранять сообщения** в компоненте **Ввод в чат**:

```
curl --request POST \
--url "http://SERVER_ADDRESS/api/v1/run/FLOW_ID" \
--header "Content-Type: application/json" \
--header "x-api-key: ASTRA_API_KEY" \
--data '{
  "input_value": "Text to input to the flow",
  "output_type": "chat",
  "input_type": "chat",
  "tweaks": {
    "ChatInput-4WKag": {
      "should_store_message": false
    }
  }
}'
```

Псевдоним ID потока

Чтобы запросы использовали псевдоним вместо реального ID потока, переименуйте эндпоинт /v1/run/\$FLOW_ID:

1. В Генеративный ИИ откройте поток, нажмите **Поделиться**, затем выберите **API-доступ**.
2. Нажмите **Схема ввода**.
3. В поле **Название эндпоинта** введите псевдоним для ID потока — запоминаемое имя. Имя может содержать только буквы, цифры, дефисы и подчёркивания, например flow-customer-database-agent.
4. Закройте панель **Схема ввода**, чтобы сохранить изменения.

Автоматически сгенерированные фрагменты кода будут использовать новое имя эндпоинта вместо исходного ID потока, например url = "http://localhost:7868/api/v1/run/flow-customer-database-agent".

Встраивание потока на сайт

Для каждого потока Генеративный ИИ предоставляет фрагмент кода, который можно вставить в <body> HTML страницы для взаимодействия с потоком через встроенный виджет чата.

Виджет чата поддерживает только потоки с компонентами **Ввод в чат** и **Вывод чата**, необходимыми для чат-опыта. Компоненты **Ввод текста** и **Текстовый вывод** могут передавать и принимать сообщения, однако они не поддерживают контекст непрерывного диалога с LLM.

Попытка взаимодействия с потоком, в котором отсутствует компонент **Ввод в чат**, запустит поток, но ответ будет содержать сообщение о пустом вводе.

Получение фрагмента кода astra-chat

Чтобы получить фрагмент кода встроенного виджета чата для потока:

1. В Генеративный ИИ откройте нужный поток.
2. Нажмите **Поделиться**, затем выберите **Встроить на сайт**.
3. Скопируйте фрагмент и вставьте его в `<body>` HTML вашей страницы. Подробнее см. в разделе Встраивание виджета чата через React, Angular или HTML.
4. Добавьте свойство `api_key`, чтобы виджет мог выполнять запросы к потоку. Подробнее см. в разделе Настройка веб-компонента astra-chat.

Виджет чата реализован как веб-компонент astra-chat, загружаемый из CDN. Подробнее см. в репозитории langflow-embedded-chat.

Например, следующий HTML встраивает виджет чата для шаблона потока **Основной промпт**, развёрнутого через ngrok:

```
<astra-chat
  host_url="https://c822-73-64-93-151.ngrok-free.app"
  flow_id="dcbcd533-859f-4b99-b1f5-16fce884f28f"
  api_key="$ASTRA_API_KEY"
></astra-chat>
```

Когда этот код размещён на живом сайте, он отображается как адаптивный чат-бот. При взаимодействии пользователя ввод запускает указанный поток, а чат-бот возвращает результат его выполнения.

Используйте CodeSandbox с встраиваемым чатом Генеративный ИИ для интерактивной демонстрации виджета на основе вашего потока. Подробнее см. в README репозитория langflow-embedded-chat.

Встраивание виджета чата через React, Angular или HTML

Ниже приведены примеры использования встроенного виджета чата в React, Angular и обычном HTML.

Чтобы использовать виджет чата в приложении React, создайте компонент, который загружает скрипт виджета и отображает интерфейс чата:

1. Объявите веб-компонент и инкапсулируйте его в React-компоненте:

```
// Объявление веб-компонента astra-chat
declare global {
  namespace JSX {
    interface IntrinsicElements {
      "astra-chat": any;
    }
  }
}

// Определение React-компонента astra-chat
export default function ChatWidget({ className }) {
  return <astra-chat></astra-chat>;
}
```

2. Разместите компонент в любом месте кода, чтобы отобразить виджет чата.

В следующем примере компонент находится в docs/src/components/ChatWidget/index.tsx. Файл index.tsx включает скрипт загрузки виджета из CDN, а также объявление и определение из предыдущего шага:

```
import React, { useEffect } from 'react';

// Компонент для загрузки скрипта виджета чата
const ChatScriptLoader = () => {
  useEffect(() => {
    if (!document.querySelector('script[src*="langflow-embedded-chat"]')) {
      const script = document.createElement('script');
      script.src = 'https://cdn.jsdelivr.net/gh/langflow-ai/langflow-embedded-chat@main/dist/build/static/js/bundle.min.js';
      script.async = true;
      document.body.appendChild(script);
    }
  }, []);

  return null;
};

// Объявление веб-компонента astra-chat
declare global {
  namespace JSX {
    interface IntrinsicElements {
      "astra-chat": any;
    }
  }
}

// Определение React-компонента astra-chat
export default function ChatWidget({ className }) {
  return <<astra-chat></astra-chat></>;
}
```

3. Импортируйте React-компонент astra-chat, чтобы сделать его доступным на странице.

Измените путь импорта, указав имя и расположение вашего компонента:

```
import ChatWidget from '@site/src/components/ChatWidget';
```

4. Вызовите компонент astra-chat в нужном месте страницы. Укажите имя вашего компонента и необходимый className:

Чтобы использовать виджет чата в приложении Angular, создайте компонент, загружающий скрипт виджета и отображающий интерфейс чата.

В Angular astra-chat — это пользовательский веб-компонент, который необходимо явно разрешить в .module.ts. Для этого добавьте CUSTOM_ELEMENTS_SCHEMA в конфигурацию модуля, а затем интегрируйте элемент <astra-chat> в шаблон.

1. В вашем приложении Angular откройте файл .module.ts, в котором хотите использовать веб-компонент astra-chat.

2. В начале файла импортируйте CUSTOM_ELEMENTS_SCHEMA:

```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
```

3. В декораторе @NgModule добавьте CUSTOM_ELEMENTS_SCHEMA в массив schemas:

```
import { NgModule, CUSTOM_ELEMENTS_SCHEMA } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppComponent } from './app.component';
```

```
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule],
```

```
schemas: [CUSTOM_ELEMENTS_SCHEMA],
providers: [],
bootstrap: [AppComponent]
})
```

```
export class AppModule { }
```

4. Откройте файл `.component.ts`, где хотите использовать виджет чата.

5. В декораторе `@Component` добавьте элемент `<astra-chat>` в ключ `template`:

```
import { Component } from '@angular/core';
```

```
@Component({
  selector: 'app-root',
  template: `
    <div class="container">
      <h1>Генеративный ИИ — тест чата</h1>
      <astra-chat
        host_url="https://c822-73-64-93-151.ngrok-free.app"
        flow_id="dcbed533-859f-4b99-b1f5-16fce884f28f"
        api_key="$ASTRA_API_KEY"
      ></astra-chat>
    </div>
  `,
  styles: [
    .container {
      padding: 20px;
      text-align: center;
    }
  ]
})
export class AppComponent {
  title = 'Astra Chat Test';
}
<html lang="en">
<head>
  <script src="https://cdn.jsdelivr.net/gh/langflow-ai/langflow-embedded-
chat@v1.0.7/dist/build/static/js/bundle.min.js"></script>
</head>
<body>
  <astra-chat
    host_url="https://c822-73-64-93-151.ngrok-free.app"
    flow_id="dcbed533-859f-4b99-b1f5-16fce884f28f"
    api_key="$ASTRA_API_KEY"
  ></astra-chat>
</body>
</html>
```

Настройка веб-компонента `astra-chat`

Для использования встроенного виджета чата веб-компонент `astra-chat` должен включать следующие обязательные параметры (`props`):

- `host_url` — URL сервера Генеративный ИИ. Должен использовать HTTPS. Не добавляйте завершающий слеш (`/`).
- `flow_id` — идентификатор встраиваемого потока.
- `api_key` — API-ключ Генеративный ИИ. Рекомендуется для предоставления виджету доступа к запуску потока.

Минимальные параметры автоматически подставляются во фрагмент кода **Встроить на сайт**, генерируемый Генеративный ИИ.

Дополнительные параметры (`props`) позволяют настраивать поведение и внешний вид виджета. Полный список параметров, их типов и описаний см. в README проекта `langflow-embedded-chat`.

Параметр `api_key`

Параметр `api_key` содержит API-ключ Генеративный ИИ, который виджет использует для аутентификации запросов.

Команда Генеративный ИИ рекомендует следовать отраслевым практикам при работе с конфиденциальными учётными данными: безопасно храните ключ и получайте его через переменную окружения.

Параметры стилей

Существует ряд параметров для настройки стиля и позиционирования виджета. Многие из них имеют тип JSON и требуют определённого форматирования в зависимости от среды встраивания.

В React и обычном HTML параметры JSON задаются как JSON-объекты или строковый JSON, например `\{"key":"value"\}`.

Для приложений Angular используйте синтаксис привязки свойств, чтобы передавать параметры JSON как JavaScript-объекты:

```
@Component({
  selector: 'app-root',
  template: `
    <div class="container">
      <h1>Тест чата</h1>
      <astra-chat
        host_url="https://c822-73-64-93-151.ngrok-free.app"
        flow_id="dcbed533-859f-4b99-b1f5-16fce884f28f"
        api_key="$ASTRA_API_KEY"
        [chat_window_style]='{"backgroundColor": "#ffffff"}'
        [bot_message_style]='{"color": "#000000"}'
        [user_message_style]='{"color": "#000000"}'
        height="600"
        width="400"
        chat_position="bottom-right"
      ></astra-chat>
    </div>
  `,
  styles: [
    .container {
      padding: 20px;
      text-align: center;
    }
  ]
})

title = 'Astra Chat Test';
}
```

Параметр `session_id`

В следующем примере добавляется пользовательский идентификатор сессии для идентификации запусков потоков, инициированных виджетом чата:

```
<astra-chat
  host_url="https://c822-73-64-93-151.ngrok-free.app"
  flow_id="dcbed533-859f-4b99-b1f5-16fce884f28f"
  api_key="$ASTRA_API_KEY"
  session_id="my-custom-session-id"
></astra-chat>
```

Параметр `tweaks`

Используйте параметр `tweaks` для изменения параметров потока во время выполнения. Доступные ключи объекта `tweaks` зависят от потока, предоставляемого через виджет.

В React и обычном HTML tweaks задаются как JSON-объект — аналогично передаче настроек в эндпоинт `/v1/run/$FLOW_ID`.

Для приложений Angular используйте синтаксис привязки свойств:

```
@Component({
  selector: 'app-root',
  template: `
    <astra-chat
      host_url="https://c822-73-64-93-151.ngrok-free.app"
      flow_id="dcbcd533-859f-4b99-b1f5-16fce884f28f"
      api_key="$ASTRA_API_KEY"
      [tweaks]='{"ChatInput-4WKag": {"should_store_message": false}}'
    ></astra-chat>
  `,
  styles: [
    .container {
      padding: 20px;
      text-align: center;
    }
  ]
})

title = 'Astra Chat Test';
}
```

Обслуживание потоков через MCP-сервер Генеративный ИИ

Каждый проект Генеративный ИИ имеет MCP-сервер, который предоставляет потоки проекта как инструменты для MCP-клиентов.

Помимо обслуживания потоков через MCP-серверы Генеративный ИИ, вы можете использовать Генеративный ИИ как MCP-клиент для доступа к любому MCP-серверу.

Взаимодействие с MCP-серверами Генеративный ИИ осуществляется через эндпоинты `/mcp API` Генеративный ИИ.

Для получения дополнительной информации см. [Использование Генеративный ИИ как MCP-сервер](#) и [Использование Генеративный ИИ как MCP-клиент](#).

Запуск потоков через совместимый с OpenAI Responses endpoint

Генеративный ИИ включает эндпоинт `/api/v1/responses`, совместимый с OpenAI Responses API, который позволяет использовать существующие клиентские библиотеки OpenAI с минимальными изменениями в коде.

Для получения дополнительной информации см. [OpenAI Responses API](#).

7.2 Запуск потоков с помощью вебхуков

Вы можете использовать компонент **Вебхук**, чтобы запускать выполнение потока в ответ на внешние события.

С помощью компонента **Вебхук** поток может получать данные напрямую из внешних источников. Затем поток может анализировать эти данные и передавать их другим компонентам для выполнения дальнейших действий, таких как вызов API, запись в базу данных или взаимодействие с LLM (Large Language Models). Если входные данные не являются корректным JSON, компонент **Вебхук** оборачивает их в объект `payload`, чтобы их можно было принять в качестве входных данных для запуска потока.

Компонент **Вебхук** предоставляет универсальную точку входа, делая потоки более событийно-ориентированными и более тесно интегрированными с остальным стеком ваших приложений и сервисов.

Например:

- Использовать LLM для анализа тональности и содержания отзывов клиентов или ответов на опросы.
- Получать уведомления от системы мониторинга и автоматически запускать ответные действия в зависимости от типа и серьезности оповещения.
- Интегрироваться с платформами электронной коммерции для обработки заказов и обновления товарных остатков.

Настройка компонента «Вебхук»

Чтобы использовать компонент **Вебхук** в потоке, выполните следующие действия:

1. В Генеративный ИИ откройте поток, в котором хотите использовать компонент **Вебхук**.
2. Добавьте в поток компонент **Вебхук** и компонент **Парсер**.
Эти два компонента обычно используют вместе, поскольку **Парсер** извлекает нужные данные из необработанной полезной нагрузки, полученной компонентом **Вебхук**.
3. Подключите выход **Данные** компонента **Вебхук** к входу **Данные** компонента **Парсер**.
4. В поле **Шаблон** компонента **Парсер** введите шаблон для преобразования необработанной полезной нагрузки в структурированный текст.

В шаблоне используйте переменные для ключей полезной нагрузки так же, как вы определяете переменные в компоненте **Шаблон промпта**.

Например, предположим, что компонент **Вебхук** получает следующие данные JSON:

```
{  
  "id": "",  
  "name": "",  
  "email": ""  
}
```

Затем вы можете использовать фигурные скобки, чтобы ссылаться на ключи JSON в любом месте шаблона парсера:

ID: {id} - Name: {name} - Email: {email}

5. Подключите выход **Распознанный текст** компонента **Парсер** к следующему логическому компоненту в потоке, например к компоненту **Ввод в чат**.

Если вы хотите протестировать только компоненты **Вебхук** и **Парсер**, можно подключить выход **Распознанный текст** напрямую к входу **Текст** компонента **Вывод чата**. После запуска потока вы сможете увидеть разобранные данные в **Playground**.

6. В поле **Конечная точка** компонента **Вебхук** скопируйте API-конечную точку, которую будете использовать для отправки данных в компонент **Вебхук** и запуска потока.

В качестве альтернативы, чтобы получить полный фрагмент кода POST /api/v1/webhook/\$FLOW_ID, откройте панель **API Access** потока и перейдите на вкладку **Webhook curl**.

Вы также можете изменить команду curl по умолчанию в поле **curl** компонента **Вебхук**. Если это поле не отображается, нажмите на компонент **Вебхук**, чтобы открыть панель инспекции компонента.

7. Отправьте POST-запрос с данными на конечную точку webhook потока, чтобы его запустить.

В следующем примере отправляется полезная нагрузка со строковыми значениями id, name и email:

```
curl -X POST "http://localhost:7860/api/v1/webhook/FLOW_ID" \  
-H "Content-Type: application/json" \  
-d '{"id": "1", "name": "John Doe", "email": "john.doe@example.com"}'
```

```
-H "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \  
-d '{"id": "12345", "name": "alex", "email": "alex@email.com"}'
```

Успешный ответ показывает, что Генеративный ИИ запустил поток.

Ответ не содержит полный вывод потока — он лишь подтверждает, что выполнение было запущено.

```
{  
  "message": "Task started in the background",  
  "status": "in progress"  
}
```

8. Чтобы просмотреть последнюю разобранную полезную нагрузку потока, нажмите на компонент **Парсер**, а затем выберите **Inspect output**.

Для приведённого выше примера разобранная полезная нагрузка будет выглядеть как строка ID: 12345 - Name: alex - Email: alex@email.com.

Запуск потоков через вебхуки Composio

Обычно компонент **Вебхук** не запускают вручную. Чтобы узнать больше о запуске потоков полезными нагрузками из внешних приложений, посмотрите видео [How to Use Webhooks in Генеративный ИИ](#).

Требование аутентификации для вебхуков

По умолчанию вебхуки выполняются от имени владельца потока без аутентификации (ASTRA_MULTIAGENT_AI_WEBHOOK_AUTH_ENABLE=False).

Чтобы требовать аутентификацию с помощью API-ключа для вызовов вебхука, установите в файле .env значение ASTRA_MULTIAGENT_AI_WEBHOOK_AUTH_ENABLE=True.

Когда аутентификация вебхуков включена, вы должны передавать API-ключ Генеративный ИИ с каждым запросом.

Чтобы передать API-ключ в виде HTTP-заголовка:

```
curl -X POST "http://SERVER_ADDRESS/api/v1/webhook/FLOW_ID" \  
-H "Content-Type: application/json" \  
-H "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \  
-d '{"id": "12345", "name": "alex", "email": "alex@email.com"}'
```

Чтобы передать API-ключ в виде параметра строки запроса:

```
curl -X POST "http://SERVER_ADDRESS/api/v1/webhook/FLOW_ID?x-api-  
key=$ASTRA_MULTIAGENT_AI_API_KEY" \  
-H "Content-Type: application/json" \  
-d '{"id": "12345", "name": "alex", "email": "alex@email.com"}'
```

Замените SERVER_ADDRESS, FLOW_ID и ASTRA_MULTIAGENT_AI_API_KEY значениями из вашего развёртывания.

Для получения дополнительной информации см. [API keys and authentication](#).

Устранение неполадок в потоках с компонентом «Вебхук»

Используйте следующую информацию, чтобы устранить распространённые проблемы, которые могут возникнуть при работе с компонентом **Вебхук**.

Проверка данных, полученных компонентом «Вебхук»

Чтобы убедиться, что компонент **Вебхук** получает данные, вы можете создать небольшой поток, который выводит только разобранную полезную нагрузку:

1. Создайте поток с компонентами **Вебхук**, **Парсер** и **Вывод чата**.
2. Подключите выход **Данные** компонента **Вебхук** к входу **Данные** компонента **Парсер**.
3. Подключите выход **Распознанный текст** компонента **Парсер** к входу **Текст** компонента **Вывод чата**.
4. В компоненте **Парсер** установите параметр **Режим** в значение **Stringify**.

В этом режиме данные, полученные компонентом **Вебхук**, передаются как строка, которую затем выводит компонент **Вывод чата**.

5. Нажмите **Поделиться**, выберите **API Access**, а затем скопируйте фрагмент кода **Webhook curl**.
6. Необязательно: измените data во фрагменте кода, если хотите передать другую полезную нагрузку.
7. Отправьте POST-запрос, чтобы запустить поток.
8. Нажмите **Playground**, чтобы убедиться, что компонент **Вывод чата** вывел JSON-данные из вашего POST-запроса.

Ошибка сборки компонента «Парсер»

Компонент **Парсер** может не собраться, если он не получает данные от компонента **Вебхук** или если со входными данными возникла проблема.

Если это происходит, попробуйте переключить **Режим** компонента **Парсер** в значение **Stringify**, чтобы компонент выводил разобранную полезную нагрузку в виде одной строки. После этого вы сможете изучить строковый вывод и устранить проблемы в шаблоне разбора либо работать с разобранными данными как со строкой.

8. Тестирование потоков

Playground в модуле «Генеративный ИИ» — это интерактивный интерфейс для тестирования потоков на основе LLM в реальном времени.

Вы можете проверять реакцию потока на различные входные данные, просматривать и изменять память, а также отслеживать вывод и логику выполнения. Например, можно убедиться, что потоки-агенты выбирают подходящие инструменты для ответа на разные запросы.

Playground позволяет итеративно дорабатывать логику и поведение потока, упрощая прототипирование приложений.

8.1 Запуск потока в Playground

Чтобы запустить поток в **Playground**, откройте поток и нажмите **Playground**. Если поток содержит компонент **Ввод в чат**, введите запрос или используйте голосовой режим, чтобы запустить поток и начать чат-сессию.

Чтобы развернуть **Playground** на весь экран, нажмите **Полный экран** в панели **Playground**.

Если в **Playground** отсутствует поле ввода сообщения, убедитесь, что поток содержит компонент **Ввод в чат**, подключённый напрямую или косвенно к порту **Ввод** компонента **Языковая модель** или **Агент**.

Поскольку **Playground** предназначен для потоков формата запрос-ответ с использованием LLM — чат-ботов и агентов, — поток должен включать компоненты **Ввод в чат**, **Языковая модель/Агент** и **Вывод чата** для полной поддержки чат-интерфейса **Playground**.

Для потоков с иным типом ввода — например, событие webhook, загрузка файла или ввод текста — вы можете запустить поток через API Генеративный ИИ, а затем открыть **Playground** для просмотра активности LLM в ходе выполнения, если применимо. Технические подробности о работе **Playground** см. в разделе Мониторинг эндпоинтов.

Просмотр логики агента

Если поток содержит компонент **Агент**, **Playground** отображает инструменты, которые использовал агент, и результат работы каждого из них. Это помогает отслеживать использование инструментов и понимать логику ответов агента. Например, следующий агент применил подключённый инструмент `fetch_content` для выполнения веб-поиска.

Просмотр истории чата

В **Playground** можно просматривать журналы сообщений для каждой чат-сессии потока, включая метки времени, содержимое и отправителей.

В боковой панели **Playground** найдите нужную чат-сессию, нажмите **Опции**, затем выберите **Журнал сообщений**.

Журнал сообщений разбивает данные типа Message для каждого сообщения чата. Нажмите любую ячейку, чтобы увидеть её полное содержимое.

Изменение памяти в Playground

Для отладки и тестирования потоков вы можете редактировать или удалять отдельные сообщения в журнале сообщений. Например, можно удалить сообщения, отправленные при тестировании компонента, который больше не является частью потока.

Чтобы удалить целую чат-сессию, в боковой панели нажмите **Опции** и выберите **Удалить**.

Изменение памяти влияет на поведение чат-бота при продолжении сессии или сохранении памяти между несколькими сессиями.

Редактирование журнала сообщений изменяет внутреннюю таблицу **messages** в модуле «Генеративный ИИ» — хранилище памяти чата по умолчанию. Подробнее об управлении сессиями и памятью чата см. в разделах Пользовательские идентификаторы сессий и Управление памятью.

8.2 Пользовательские идентификаторы сессий

Чат-сессии идентифицируются по идентификатору сессии (**session_id**) — уникальному идентификатору выполнения потока.

По умолчанию идентификатор сессии совпадает с идентификатором потока: все сообщения чата хранятся под одним **session_id** как единая сессия.

Если нужно сохранять контекст между несколькими запусками потока или разделять сессии при отладке, задайте пользовательский **session_id**.

Пользовательские идентификаторы сессий полезны в следующих случаях:

- Разделение чат-сессий, когда один поток обслуживает несколько одновременных взаимодействий — например, чат-бот с несколькими параллельными пользователями.
- Сохранение памяти при продолжении сессии через несколько запусков потока или при передаче контекста между потоками.
- Разграничение активности нескольких пользователей в рамках одного потока.
- Идентификация собственных чат-сессий при отладке и тестировании.

Пользовательские идентификаторы сессий можно задавать в визуальном редакторе и программно через API.

В компонентах ввода и вывода используйте поле **Идентификатор сеанса**:

1. Нажмите на компонент, в котором хотите задать пользовательский идентификатор сессии.
2. В панели инспекции компонента включите **Идентификатор сеанса**.
3. Нажмите **Закрыть**.
4. Введите пользовательский идентификатор сессии. Если поле пусто, поток использует идентификатор по умолчанию.
5. Откройте **Playground**, чтобы начать чат с заданным идентификатором сессии.

Не забудьте изменить **Идентификатор сеанса**, когда хотите начать новую сессию или продолжить предыдущую с другим идентификатором.

При запуске потока через API Генеративный ИИ включите параметр **session_id** в тело запроса. Например:

```
curl -X POST "http://$LANGFLOW_SERVER_ADDRESS/api/v1/run/$FLOW_ID" \  
-H "Content-Type: application/json" \  
-H "x-api-key: $ASTRA_API_KEY" \  
-d '{  
  "session_id": "CUSTOM_SESSION_ID",  
  "input_value": "message",  
  "input_type": "chat",  
  "output_type": "chat"  
}'
```

Эта команда запускает новую чат-сессию с указанным `session_id` или продолжает существующую сессию с таким идентификатором.

В рабочей среде рекомендуется использовать переменную для идентификатора сессии вместо жёстко заданного значения.

Например, если нужно сохранять контекст для аутентифицированных пользователей, в качестве идентификатора сессии удобно использовать идентификатор пользователя. Если каждый чат должен быть уникальным, генерируйте новый UUID для каждой сессии автоматически. Подробнее см. в разделе Идентификатор сеанса.

8.3 Совместный доступ к Playground потока

Публичный Playground предназначен только для тестирования и не предназначен для встраивания потоков в приложения. Для информации о запуске потоков в приложениях и на сайтах см. Запуск потоков через API Генеративный ИИ.

Публичный Playground недоступен в модуле «Генеративный ИИ» Desktop. Опция **Публичный Playground** открывает **Playground** для отдельного потока по эндпоинту `/public_flow/$FLOW_ID`.

После развёртывания публичного сервера Генеративный ИИ вы можете поделиться этим URL с другим пользователем, чтобы предоставить ему доступ только к **Playground** указанного потока. Пользователь может взаимодействовать с вводом и выводом чата и просматривать результаты — без необходимости устанавливать Генеративный ИИ или создавать API-ключ.

Чтобы поделиться **Playground** потока:

1. В Генеративный ИИ откройте нужный поток.
2. В рабочем пространстве нажмите **Поделиться**, затем включите **Публичный Playground**.
3. Снова нажмите **Публичный Playground**, чтобы открыть окно **Playground**. URL этого окна — адрес публичного **Playground** потока, например:

`https://3f7c-73-64-93-151.ngrok-free.app/playground/d764c4b8-5cec-4c0f-9de0-4b419b11901a`

4. Отправьте URL пользователю, которому хотите предоставить доступ к **Playground** потока.

9. Импорт и экспорт

Вы можете экспортировать потоки для переноса между экземплярами Генеративный ИИ, обмена с другими пользователями или создания резервных копий.

9.1 Экспорт потока

Доступны три способа экспортировать поток:

- **Экспорт из проектов:** на странице **Проекты** найдите нужный поток, нажмите **Ещё**, затем выберите **Экспорт**. Чтобы экспортировать все потоки проекта, нажмите **Опции** в списке **Проекты** и выберите **Скачать**.
- **Экспорт через меню «Поделиться»:** при редактировании потока нажмите **Поделиться**, затем **Экспорт**.
- **Экспорт через API Генеративный ИИ:** для экспорта одного потока используйте эндпоинт `/flows/download`; для экспорта всех потоков проекта — эндпоинт `/projects/download`.

Экспортированные потоки загружаются на локальный компьютер в виде JSON-файлов с именем `FLOW_NAME.json`. При экспорте целого проекта JSON-файлы упаковываются в zip-архив. Подробнее о структуре файлов см. в разделе **Содержимое JSON-файла Генеративный ИИ**.

Параметр «Сохранить с моими API-ключами»

При экспорте со страницы **Проекты** или из меню **Поделиться** можно включить параметр **Сохранить с моими API-ключами**, чтобы экспортировать поток вместе с определёнными переменными API-ключей. Переменные, не являющиеся API-ключами, включаются в экспорт независимо от этого параметра.

Если вы вводите ключ напрямую в поле API-ключа компонента, параметр **Сохранить с моими API-ключами** экспортирует его буквальное значение.

Если ключ хранится в глобальной переменной Генеративный ИИ, параметр **Сохранить с моими API-ключами** экспортирует только имя переменной. При импорте потока в другой экземпляр Генеративный ИИ в нём должны быть глобальные переменные с теми же именами и действительными значениями — иначе поток не выполнится.

Если какие-либо переменные отсутствуют или недействительны, их необходимо создать или отредактировать после импорта.

9.2 Импорт потока

Вы можете импортировать JSON-файлы Генеративный ИИ с локального компьютера следующими способами:

- **Импорт в проекты:** на странице **Проекты** нажмите **Загрузить поток** и выберите JSON-файл для импорта.
- **Импорт перетаскиванием:** перетащите JSON-файлы из файлового менеджера в окно Генеративный ИИ — это работает на любой странице.
- **Импорт через API Генеративный ИИ:** для импорта одного JSON-файла используйте эндпоинт `/flows/upload/`; для импорта zip-архива с JSON-файлами — эндпоинт `/projects/upload`.

Запуск импортированного потока

После импорта поток готов к использованию. Если в нём есть глобальные переменные, убедитесь, что в вашем экземпляре Генеративный ИИ есть переменные с теми же именами и действительными значениями. Подробнее см. в разделе Параметр «Сохранить с моими API-ключами».

9.3 Содержимое JSON-файла Генеративный ИИ

Экспортированный поток загружается в виде JSON-файла с именем FLOW_NAME.json. JSON-файлы Генеративный ИИ содержат следующее:

- Узлы и рёбра — описывают компоненты и соединения в потоке.
- Дополнительные метаданные — описывают поток и проект, к которому он относится.

Примеры JSON-файлов Генеративный ИИ можно найти среди шаблонов в репозитории Генеративный ИИ. Также можно создать поток из шаблона, экспортировать его и открыть в текстовом редакторе.

Узлы

Узлы представляют компоненты, из которых состоит поток. Например, следующий объект представляет компонент **Ввод в чат**:

```
{
  "data": {
    "description": "Get chat inputs from the Playground.",
    "display_name": "Chat Input",
    "id": "ChatInput-jFwUm",
    "node": {
      "base_classes": ["Message"],
      "description": "Get chat inputs from the Playground.",
      "display_name": "Chat Input",
      "icon": "MessagesSquare",
      "template": {
        "input_value": {
          "display_name": "Text",
          "info": "Message to be passed as input.",
          "value": "Hello"
        },
        "sender": {
          "value": "User",
          "options": ["Machine", "User"]
        },
        "sender_name": {
          "value": "User"
        },
        "should_store_message": {
          "value": true
        }
      }
    },
    "type": "ChatInput"
  },
  "position": {
    "x": 689.5720422421635,
    "y": 765.155834131403
  }
}
```

Каждый узел имеет уникальный идентификатор в формате NODE_NAME-UUID, например ChatInput-jFwUm.

Узлы точек входа, такие как ChatInput, выполняются первыми при запуске потока.

Рёбра

Рёбра представляют соединения между узлами.

Следующий пример описывает ребро (соединение) между узлами ChatInput и OpenAIModel:

```
{
  "className": "",
  "data": {
    "sourceHandle": {
      "dataType": "ChatInput",
      "id": "ChatInput-jFwUm",
      "name": "message",
      "output_types": ["Message"]
    },
    "targetHandle": {
      "fieldName": "input_value",
      "id": "OpenAIModel-OcXkl",
      "inputTypes": ["Message"],
      "type": "str"
    }
  },
  "id": "reactflow__edge-ChatInput-jFwUm{...}-OpenAIModel-OcXkl{...}",
  "source": "ChatInput-jFwUm",
  "sourceHandle": "{\u0391data\u0391type\u0391: \u0391ChatInput\u0391, \u0391id\u0391: \u0391ChatInput-jFwUm\u0391, \u0391name\u0391: \u0391message\u0391, \u0391output_types\u0391: [\u0391Message\u0391]}",
  "target": "OpenAIModel-OcXkl",
  "targetHandle": "{\u0391fieldName\u0391: \u0391input_value\u0391, \u0391id\u0391: \u0391OpenAIModel-OcXkl\u0391, \u0391inputTypes\u0391: [\u0391Message\u0391], \u0391type\u0391: \u0391str\u0391}"
}
```

Это ребро показывает, что компонент ChatInput выводит тип Message в целевой узел OpenAIModel. Компонент OpenAIModel принимает тип Message в поле input_value.

Дополнительные метаданные и информация о проекте

Дополнительная информация о потоке хранится в корневом объекте data:

- Метаданные и информация о проекте: название, описание и last_tested_version.

Например:

```
{
  "name": "Basic Prompting",
  "description": "Perform basic prompting with an OpenAI model.",
  "tags": ["chatbots"],
  "id": "1511c230-d446-43a7-bfc3-539e69ce05b8",
  "last_tested_version": "1.0.19.post2",
  "gradient": "2",
  "icon": "Braces"
}
```

- Визуальная информация о потоке: определяет позицию области просмотра при открытии потока в рабочем пространстве:

```
"viewport": {
  "x": -37.61270157375441,
  "y": -155.91266341888854,
  "zoom": 0.7575251406952855
}
```

- Заметки — комментарии, поясняющие назначение потока, детали конфигурации и другую полезную информацию. Могут содержать текст, ссылки, фрагменты кода и т. д. Заметки кодируются в формате Markdown и хранятся как объекты node.

```
{
  "id": "undefined-kVLkG",
  "node": {
```

```
"description": "## 📖 README\nPerform basic prompting with an OpenAI model.\n\n#### Quick Start\n- Add your\n**OpenAI API key** to the **OpenAI Model**\n- Open the **Playground** to chat with your bot.\n..."
}
```

10. Использование агентов

Компонент **Агент** Генеративный ИИ критически важен для построения потоков агентов. Этот компонент предоставляет всё необходимое для создания агента, включая несколько поставщиков Больших Языковых моделей (LLM), вызов инструментов и пользовательские инструкции. Он упрощает конфигурацию агента, позволяя сосредоточиться на разработке приложений.

Как работают агенты?

Агенты расширяют Большие Языковые модели (LLM), интегрируя *инструменты*, которые представляют собой функции, предоставляющие дополнительный контекст и позволяющие автономно выполнять задачи. Эти интеграции делают агентов более специализированными и мощными, чем отдельные LLM.

В то время как LLM может генерировать приемлемые, но пассивные ответы на общие запросы и задачи, агент может использовать интегрированный контекст и инструменты, чтобы предоставлять более релевантные ответы и даже предпринимать действия. Например, вы можете создать агента, который получает доступ к документации вашей компании, репозиториям и другим ресурсам, помогая вашей команде с задачами, требующими знаний о ваших конкретных продуктах, клиентах и коде.

Агенты используют LLM в качестве движка рассуждений для обработки ввода, определения, какие действия следует выполнить для ответа на запрос, и затем генерируют ответ. Ответ может быть типичным ответом LLM на основе текста, либо включать действие, например редактирование файла, запуск скрипта или вызов внешнего API.

В контексте агента инструменты — это функции, которые агент может выполнять для выполнения задач или доступа к внешним ресурсам. Функция обернута в объект Инструмент с общим интерфейсом, понятным агенту. Агенты узнают о инструментах через регистрацию инструментов, когда агенту предоставляется список доступных инструментов, обычно при инициализации агента. Описание объекта Инструмент сообщает агенту, что может делать инструмент, чтобы он мог решить, подходит ли инструмент для конкретного запроса.

10.1 Использование компонента Агент в потоке

Следующие шаги объясняют, как создать поток агента в модуле «Генеративный ИИ», начиная с пустого потока. Для готового примера используйте шаблон **Простой Агент** или Генеративный ИИ быстрый старт.

1. Нажмите **Новый Поток**, затем нажмите **Пустой Поток**.
2. Добавьте компонент **Агент** в ваш поток.
3. Чтобы изменить глобальную конфигурацию поставщика моделей в модуле «Генеративный ИИ», выполните следующее:
 1. Чтобы открыть панель **Поставщики Модели**, нажмите на значок профиля, выберите **Настройки**, затем нажмите **Поставщики Модели**.
 2. В панели **Поставщики Модели** выберите поставщика.
 3. В поле **Ключ API** введите API-ключ вашего поставщика.
Ключ должен иметь разрешение на вызов моделей, которые вы хотите использовать в потоке, и ваш аккаунт должен иметь достаточное количество кредитов для выполнения требуемых действий.

Для каждого поставщика можно добавить только один ключ. Убедитесь, что ключ имеет доступ ко *всем* моделям, которые вы хотите использовать в модуле «Генеративный ИИ».

4. Включите конкретные модели, которые вы хотите использовать в модуле «Генеративный ИИ». Доступные модели зависят от поставщика и разрешений вашего API-ключа. Модели, генерирующие Текст, перечислены в разделе **Языковые модели**. Модели, генерирующие embeddings, перечислены в разделе **Модели Эмбедингов**.

После включения модели в глобальной конфигурации моделей Генеративный ИИ, вы сможете использовать эту модель в любом компоненте, основанном на модели, в ваших потоках.

4. Выберите модель, которую хотите использовать, из выпадающего списка **Языковая модель**. Если желаемая модель не указана в списке, убедитесь, что она включена в конфигурации **Модели**. Для получения дополнительной информации см. параметры компонента **Агент**.

5. Добавьте компоненты **Ввод в чат** и **Вывод чата** в ваш поток, а затем подключите их к компоненту **Агент**.

На данном этапе вы создали базовый чат-поток на основе LLM, который можно протестировать в **Playground**. Однако этот поток общается только с LLM. Чтобы улучшить этот поток и сделать его действительно агентным, добавьте несколько инструментов, как объясняется в следующих шагах.

6. Добавьте компоненты **Веб-поиск**, **URL** и **Калькулятор** в ваш поток.

7. Включите **Режим Инструмента** в компонентах **Веб-поиск**, **URL** и **Калькулятор**:

1. Нажмите на компонент **Веб-поиск**, чтобы открыть меню заголовка компонента, затем включите **Режим Инструмента**.
2. Повторите действие для компонентов **URL** и **Калькулятор**.
3. Подключите порт **Набора инструментов** каждого инструмента к порту **Инструменты** на компоненте **Агент**.

Режим Инструмента превращает компонент в инструмент, изменяя входные данные компонента. При включённом **Режиме Инструмента** компонент может принимать запросы от компонента **Агент**, используя доступные действия компонента в качестве инструментов. В режиме **Режим Инструмента** у компонента есть порт **Набора инструментов**, который необходимо подключить к порту **Инструменты** компонента **Агент**, если вы хотите позволить агенту использовать действия этого компонента в качестве инструментов.

Для получения дополнительной информации см. **Настройка инструментов для агентов**.

8. Откройте **Playground**, затем задайте агенту **Какие инструменты ты используешь для ответа на мой вопрос?**

Агент должен ответить списком подключённых инструментов. Он также может включать встроенные инструменты.

Для ответа на ваши вопросы я использую сочетание своих собственных знаний (по состоянию на июнь 2024 года) и набора внешних инструментов. Вот основные типы инструментов, которые я могу использовать:

Поиск в интернете и получение контента: я могу получать и обобщать контент с веб-страниц, включая рекурсивное сканирование ссылок.

Поиск новостей: я могу искать свежие новостные статьи с помощью Google News через RSS-каналы.

Калькулятор: я могу выполнять арифметические вычисления и оценивать математические выражения.

Дата и время: я могу предоставить текущую дату и время в различных часовых поясах.

Эти инструменты помогают мне предоставлять актуальную информацию, выполнять вычисления и получать

необходимые данные из интернета. Если у вас есть конкретный вопрос, сообщите мне, и я использую наиболее подходящий инструмент (или инструменты), чтобы помочь!

9. Чтобы протестировать конкретный инструмент, задайте агенту вопрос, использующий один из инструментов, например **Обобщи сегодняшние новости**.

Чтобы помочь вам отлаживать и тестировать ваши потоки, **Playground** отображает вызовы инструментов агента, предоставленный ввод и необработанный вывод, полученный агентом перед генерацией обобщения. В данном примере агент должен вызвать компонент **Веб-поиск с Режим поиска**, установленным в **Новости**.

Вы успешно создали базовый поток агента, использующий некоторые общие инструменты.

Чтобы продолжить работу над этим руководством, попробуйте подключить другие компоненты инструментов или использовать Генеративный ИИ в качестве MCP-клиента для поддержки более сложных и специализированных задач.

Для примера с несколькими агентами см. [Использовать агент в качестве инструмента](#).

10.2 Параметры компонента Агент

Вы можете настроить компонент **Агент**, чтобы он использовал вашего провайдера и модель, пользовательские инструкции и инструменты.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонентов, которая появляется при выборе компонента.

Провайдер и модель

Используйте настройку **Языковая модель** (agent_llm), чтобы выбрать LLM, которую агент будет использовать.

Чтобы изменить глобальную конфигурацию провайдера модели в модуле «Генеративный ИИ», выполните следующее:

1. Чтобы открыть панель **Поставщики Модели**, нажмите на значок профиля, выберите **Настройки**, а затем нажмите **Поставщики Модели**.
2. В панели **Поставщики Модели** выберите провайдера.
3. В поле **Ключ API** добавьте API-ключ вашего провайдера.

Ключ должен иметь разрешение на вызов моделей, которые вы хотите использовать в вашем потоке, и ваш аккаунт должен иметь достаточное количество кредитов для выполнения требуемых действий.

Для каждого провайдера можно добавить только один ключ. Убедитесь, что ключ имеет доступ ко *всем* моделям, которые вы хотите использовать в модуле «Генеративный ИИ».

4. Включите конкретные модели, которые вы хотите использовать в модуле «Генеративный ИИ». Доступные модели зависят от провайдера и прав вашего API-ключа. Модели, генерирующие Текст, перечислены в разделе **Языковые модели**. Модели, генерирующие эмбединги, перечислены в разделе **Модели Эмбедингов**.

После того как вы включите модель в глобальной конфигурации модели Генеративный ИИ, вы сможете использовать её в любом компоненте, основанном на модели, в ваших потоках.

Чтобы использовать модель с компонентом **Агент**, выберите модель в поле **Языковая модель** компонента **Агент**.

Поле **Языковая модель** отображает все языковые модели, которые вы настроили глобально. Если у провайдера нет доступных языковых моделей, они не отображаются. Например, если провайдер предлагает только модели эмбедингов, эти модели не отображаются в компоненте **Агент**.

Чтобы получить доступ к другим провайдерам или моделям, вы можете выполнить одно из следующих действий:

- Подключите любой компонент языковой модели к порту **Языковая модель** компонента **Агент**. Этот вариант позволяет подключить пользовательский компонент языковой модели для использования моделей, недоступных в глобальном списке провайдеров моделей.
- Настройте дополнительные провайдеры в панели **Модели**, а затем выберите модель из выпадающего списка **Языковая модель**.

Если вам необходимо генерировать эмбединги в вашем потоке, используйте компонент модели эмбедингов.

Агент инструкции и ввод

В поле **Инструкции агента** (system_prompt) вы можете задать пользовательские инструкции, которые компонент **Агент** будет использовать в каждом разговоре.

Эти инструкции применяются в дополнение к **Ввод** (input_value), которые могут быть введены напрямую или предоставлены через другой компонент, например, компонент **Ввод в чат**.

Инструменты

Агенты наиболее полезны, когда у них есть соответствующие инструменты для выполнения запросов.

Компонент **Агент** может использовать любой компонент Генеративный ИИ в качестве инструмента, включая других агентов и серверы MCP.

Чтобы присоединить компонент в качестве инструмента, необходимо включить **Режим Инструмента** на компоненте, который вы хотите присоединить, а затем подключить его к порту **Инструменты** компонента **Агент**. Для получения дополнительной информации см. настройка инструментов агентов.

Чтобы позволить агентам использовать инструменты с серверов MCP, используйте компонент **Инструменты MCP**.

Память агента

Агенты Генеративный ИИ имеют встроенную память чата, которая включена по умолчанию. Эта память позволяет им извлекать и ссылаться на сообщения из предыдущих разговоров, поддерживая скользящее окно контекста для каждого идентификатора сеанса чата.

Память чата группируется по ID сеанса (session_id). Рекомендуется использовать пользовательские ID сеанса, если необходимо разделять память чата для разных пользователей или приложений, которые запускают один и тот же поток.

По умолчанию компонент **Агент** использует хранилище вашей установки Генеративный ИИ и извлекает ограниченное количество сообщений чата, которое можно настроить с помощью параметра **Количество сообщений в истории чата**.

Компонент **История сообщений** не требуется для памяти чата по умолчанию, но необходим, если вы хотите использовать внешнюю память чата, например Mem0. Кроме того, компонент **История сообщений** предоставляет дополнительные возможности сортировки,

фильтрации и ограничения памяти. Однако большинство этих опций встроены в компонент **Агент** со значениями по умолчанию.

Для получения дополнительной информации см. хранение памяти чата и компонент **История сообщений**.

Дополнительные параметры

В компоненте **Агент** доступные параметры могут изменяться в зависимости от выбранного провайдера и модели, включая поддержку дополнительных режимов, аргументов или функций, таких как память чата и температура. Например:

- **Текущая дата** (add_current_date_tool): При включении (true) эта настройка добавляет инструмент агенту, который может получить текущую дату.
- **Обработка ошибок разбора** (handle_parsing_errors): При включении (true) эта настройка позволяет агенту исправлять ошибки, такие как опечатки, при анализе ввода пользователя.
- **Подробный вывод** (verbose): При включении (true) эта настройка записывает подробный журнал для отладки и анализа.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

10.3 Вывод компонента Агент

Компонент **Агент** выводит **Ответ** (response), который представляет собой данные Message , содержащие необработанный ответ агента на запрос. Обычно это передаётся в компонент **Вывод чата**, чтобы вернуть ответ в формате Системное сообщение. Это также может быть передано другим компонентам, если необходимо дополнительно обработать ответ перед возвратом пользователю или вместе с ним.

11. Настройка инструментов

По умолчанию агенты Генеративный ИИ включают только функциональность, встроенную в их базовую LLM.

Вы можете присоединять инструменты к агентам, чтобы предоставить доступ к дополнительной, целевой функциональности. Например, инструменты могут использоваться для создания специализированных агентов, таких как агент поддержки клиентов, который может получать доступ к базе знаний компании, финансовый агент, который может получать цены акций, или агент-репетитор по математике, который может использовать продвинутые математические функции для решения сложных уравнений.

11.1 Присоединение инструментов

Чтобы присоединить инструмент к агенту, соедините выход **Инструмент** любого компонента с входом **Инструмент** компонента **Агент**.

Некоторые компоненты по умолчанию генерируют выход **Инструмент**. Для всех остальных компонентов необходимо включить **Режим Инструмента** в меню заголовка компонента. Затем вы можете подключить инструмент к агенту.

Вы можете подключить несколько инструментов к одному агенту, и каждый инструмент может иметь несколько действий (функций), которые агент может вызывать.

Когда вы запускаете ваш поток, агент решает, когда использовать определённые инструменты, если он определит, что инструмент может помочь ему ответить на запрос пользователя.

Редактирование действий инструмента

Когда вы присоединяете компоненты к агенту в качестве инструментов, каждый инструмент может иметь несколько действий (функций), которые агент может вызывать. Доступные действия перечислены в списке **Действия** каждого компонента инструмента.

Вы можете изменить метки, описания и доступность каждого действия, чтобы помочь агенту понять, как использовать инструмент, и предотвратить использование нерелевантных или нежелательных действий.

Если агент, кажется, использует инструмент неправильно, попробуйте отредактировать метаданные действий, чтобы уточнить назначение инструмента и отключить ненужные действия.

Вы также можете попробовать использовать компонент **Шаблон промпта**, чтобы передать агенту дополнительные инструкции или примеры. Чтобы просмотреть и отредактировать действия инструмента, нажмите **Настройка Действий Инструмента** на компоненте инструмента.

Для каждого действия предоставляется следующая информация:

- **Enabled:** Флажок, определяющий, доступно ли действие агенту. Если выбран, действие включено. Если не выбран, действие отключено.
- **Название:** Строковое имя Системное сообщение для действия, например Получить контент. Это нельзя изменить.
- **Описание:** Системное сообщение описание назначения действия, например Загрузка контента с веб-страниц. Чтобы изменить это значение, дважды щелкните строку

действия, чтобы открыть панель редактирования. Изменения сохраняются автоматически при выходе из поля или закрытии диалогового окна.

- **Slug:** Закодированное имя действия, обычно совпадающее с именем, например Получить_контент. Чтобы изменить это значение, дважды щелкните строку действия, чтобы открыть панель редактирования. Изменения сохраняются автоматически при выходе из поля или закрытии диалогового окна.

Некоторые действия позволяют задать фиксированные значения для их входов. Обычно рекомендуется оставлять их пустыми, чтобы агент мог предоставить свои собственные значения. Однако вы можете использовать фиксированное значение, если пытаетесь отладить поведение агента или ваш сценарий требует фиксированного ввода для действия.

11.2 Использование агента в качестве инструмента

Чтобы создать мультиагентные потоки, вы можете установить другой компонент **Агент** в режим **Режим Инструмента**, а затем присоединить этого агента в качестве инструмента к вашему основному компоненту **Агент**.

Чтобы попробовать это самостоятельно, добавьте дополнительного агента в шаблон **Простой Агент**:

1. Создайте поток на основе шаблона **Простой Агент**.
2. Добавьте второй компонент **Агент** в поток.
3. Добавьте ваш **API-ключ OpenAI** к обоим компонентам **Агент**.
4. Во втором компоненте **Агент** измените модель на gpt-4.1, а затем включите **Режим Инструмента**.
5. Нажмите **Настройка Действий Инструмента**, чтобы отредактировать действия инструмента.

Для этого примера измените slug действия на Agent-gpt-41 и задайте описание Используйте модель gpt-4.1 для решения задач. Это позволяет основному агенту знать, что этот инструмент использует модель gpt-4.1, что может быть полезно для задач, требующих более широкого контекстного окна, таких как крупные задачи скрейпинга и поиска.

В качестве другого примера вы можете присоединить несколько специализированных моделей к основному агенту, например, агентов, обученных на определённых задачах или в определённых доменах, и тогда основной агент будет вызывать каждый специализированный агент по мере необходимости для ответа на запросы.

Вы также можете включать и отключать инструменты, если хотите ограничить доступный набор инструментов.

6. Подключите порт **Набора Инструментов** нового агента к порту **Инструменты** существующего агента.

11.3 Добавление пользовательских компонентов в качестве инструментов

Агент может использовать пользовательские компоненты в качестве инструментов.

1. Чтобы добавить пользовательский компонент в поток агента, нажмите **Новый Пользовательский компонент** в меню **Основные компоненты** или **Пакеты**.
2. Введите код Python в панель **Код**, чтобы создать пользовательский компонент.

Если у вас ещё нет кода для пользовательского компонента, вы можете использовать следующий фрагмент кода в качестве примера перед созданием собственного.

Кастомный компонент по анализу текста

Этот код создает компонент анализатора текста.

```
from astra.custom import Component
from astra.io import MessageTextInput, Output
from astra.schema import Data
import re

class TextAnalyzerComponent(Component):
    display_name = "Text Analyzer"
    description = "Analyzes and transforms input text."
    documentation: str = "http://docs.langflow.org/components/custom"
    icon = "chart-bar"
    name = "TextAnalyzerComponent"

    inputs = [
        MessageTextInput(
            name="input_text",
            display_name="Input Text",
            info="Enter text to analyze",
            value="Hello, World!",
            tool_mode=True,
        ),
    ]

    outputs = [
        Output(display_name="Analysis Result", name="output", method="analyze_text"),
    ]

    def analyze_text(self) -> Data:
        text = self.input_text

        # Perform text analysis
        word_count = len(text.split())
        char_count = len(text)
        sentence_count = len(re.findall(r'\w+[\.!?', text))

        # Transform text
        reversed_text = text[::-1]
        uppercase_text = text.upper()

        analysis_result = {
            "original_text": text,
            "word_count": word_count,
            "character_count": char_count,
            "sentence_count": sentence_count,
            "reversed_text": reversed_text,
            "uppercase_text": uppercase_text
        }

        data = Data(value=analysis_result)
        self.status = data
        return data
```

3. Включите **Режим Инструмента** в пользовательском компоненте.
4. Подключите выход инструмента пользовательского компонента к входу **Инструменты** компонента **Агент**.
5. Откройте **Playground** и дайте агенту инструкцию, Примените анализатор текста к следующему сообщению: «Агенты действительно мыслящие машины!»

В соответствии с вашей инструкцией агент должен вызвать действие `analyze_text` и вернуть результат.

11.4 Сделай любой компонент инструментом

Если у компонента, который вы хотите использовать в качестве инструмента, нет кнопки **Режим Инструмента**, добавьте `tool_mode=True` к одному из входов компонента и подключите новый выход **Набор Инструментов** к входу **Инструменты** агента.

Генеративный ИИ поддерживает **Режим Инструмента** для следующих типов данных:

- `DataInput`
- `DataFrameInput`
- `PromptInput`
- `MessageTextInput`
- `MultilineInput`
- `DropdownInput`

Например, пример кода в Использовании пользовательских компонентов в качестве инструментов, включающий `tool_mode=True` на вход `MessageTextInput`, чтобы пользовательский компонент мог использоваться в качестве инструмента:

```
inputs = [  
    MessageTextInput(  
        name="input_text",  
        display_name="Input Text",  
        info="Enter text to analyze",  
        value="Hello, World!",  
        tool_mode=True,  
    ),  
]
```

11.5 Использование потоков как инструменты

Агент может использовать ваши другие потоки в качестве инструментов с помощью компонента **Запустить поток**.

1. Добавьте компонент **Запустить поток** в ваш поток.
2. Выберите поток, который агент будет использовать как инструмент.
3. Включите **Режим Инструмента**. Выбранный поток становится действием в компоненте **Запустить поток**.
4. Подключите выход **Инструмент** компонента **Запустить поток** к входу **Инструменты** компонента **Агент**.
5. Откройте **Playground**, а затем задайте агенту Какие инструменты вы используете для ответа на мои вопросы?. Ваш поток должен быть виден в ответе как доступный инструмент.
6. Задайте агенту вопрос, который явно использует подключенный поток в качестве инструмента. Подключенный поток возвращает ответ на основе вашего вопроса.

12. MCP клиент

Генеративный ИИ интегрируется с MCP (Model Context Protocol) как в роли сервера MCP, так и в роли клиента MCP.

Эта страница описывает использование Генеративный ИИ в роли клиента MCP с помощью компонента **Инструменты MCP** и подключённых серверов MCP.

Для получения информации об использовании Генеративный ИИ в роли сервера MCP см. Генеративный ИИ в роли сервера MCP.

12.1 Использование компонента «Инструменты MCP»

Компонент **Инструменты MCP** подключается к серверу MCP, чтобы агент Генеративный ИИ мог использовать инструменты сервера при ответах на запросы пользователей.

Компонент работает в двух режимах в зависимости от типа сервера:

- Подключение к стороннему серверу MCP — через JSON-файл конфигурации, команду запуска или URL-адрес HTTP/SSE для доступа к инструментам внешних серверов MCP.
- Подключение к серверу MCP Генеративный ИИ — для использования потоков из ваших проектов Генеративный ИИ в качестве инструментов MCP.

Подключение к стороннему серверу MCP

`uvx` включён в пакет `uv`, поставляемый вместе с модулем «Генеративный ИИ».

Для использования команд сервера на основе `prx` сначала установите LTS-версию Node.js. Если вы запускаете Генеративный ИИ в Docker, установите Node.js внутри образа контейнера и пересоберите его, чтобы серверы MCP на основе `prx` были доступны во время выполнения. Для получения дополнительной информации см. Управление пакетами.

Пример сервера MCP на основе `prx` в модуле «Генеративный ИИ» см. в разделе Подключение сервера Astra DB MCP к модулю «Генеративный ИИ». 1. Добавьте компонент **Инструменты MCP** в ваш поток.

2. В поле **Сервер MCP** выберите ранее подключённый сервер или нажмите **Добавить сервер MCP**.

Существует несколько способов добавить новый сервер:

- **JSON** — вставьте объект JSON конфигурации сервера MCP в поле, включая нужные параметры, и нажмите **Добавить сервер**.
- **STDIO** — введите **Название**, **Команду** и при необходимости **Аргументы** и **Переменные окружения**, затем нажмите **Добавить сервер**. Например, чтобы запустить сервер Fetch, в поле **Команда** укажите `uvx mcp-server-fetch`.
- **HTTP/SSE** — введите **Название**, **URL** и при необходимости **Заголовки** и **Переменные окружения**, затем нажмите **Добавить сервер**. URL по умолчанию для серверов MCP Генеративный ИИ: `http://localhost:7860/api/v1/mcp/project/PROJECT_ID/streamable` или `http://localhost:7860/api/v1/mcp/streamable`. Для получения дополнительной информации см. Подключение к серверу MCP Генеративный ИИ.

3. Чтобы настроить заголовки для сервера MCP, введите каждый заголовок в поля **Заголовки** в виде пар «ключ — значение». В значениях заголовков можно использовать глобальные переменные, указав имя глобальной переменной в качестве

значения. Для получения дополнительной информации см. Использование глобальных переменных в заголовках сервера MCP.

4. Чтобы использовать переменные окружения в команде сервера, введите каждую переменную в поля **Env** в виде пар «ключ — значение».
5. В поле **Инструмент** выберите инструмент для использования в этом компоненте или оставьте поле пустым, чтобы предоставить доступ ко всем инструментам сервера MCP. Если вы выбираете конкретный инструмент, может потребоваться настройка дополнительных полей. Для получения информации о полях, специфичных для инструмента, см. документацию вашего сервера MCP.

На данном этапе компонент **Инструменты MCP** предоставляет инструмент с подключённого сервера, но ни один компонент его не использует. Следующие шаги объясняют, как сделать инструмент доступным для компонента **Агент**.

6. В меню заголовка компонента включите **Режим инструмента**, чтобы использовать компонент с агентом.
7. Подключите порт **Toolset** компонента **Инструменты MCP** к порту **Инструменты** компонента **Агент**.
Убедитесь, что к компоненту **Агент** также подключены компоненты **Ввод в чат** и **Вывод чата**.
8. Протестируйте поток, чтобы убедиться, что сервер MCP подключён и агент использует выбранный инструмент. Откройте **Playground** и введите запрос, задействующий инструмент из компонента **Инструменты MCP**.

Например, если вы используете mcp-server-fetch с инструментом fetch, можно попросить агента подвести итоги последних новостей в сфере технологий. Агент вызовет функцию fetch сервера MCP и вернёт ответ.

9. Если вы хотите, чтобы агент мог использовать больше инструментов, повторите шаги выше, добавив дополнительные компоненты инструментов с другими серверами или инструментами.

Подключение к серверу MCP Генеративный ИИ

Каждый проект Генеративный ИИ запускает отдельный сервер MCP, который предоставляет потоки проекта в виде инструментов MCP. Для получения дополнительной информации о серверах MCP ваших проектов, включая публикацию потоков как инструментов MCP, см. Генеративный ИИ в роли сервера MCP.

Серверы MCP Генеративный ИИ поддерживают транспорт **Streamable HTTP**, а также **Server-Sent Events (SSE)** в качестве резервного варианта.

Чтобы использовать потоки как инструменты, подключитесь к конечной точке MCP проекта через компонент **Инструменты MCP**:

1. Добавьте компонент **Инструменты MCP** в ваш поток, нажмите **Добавить сервер MCP** и выберите режим **HTTP/SSE**.
2. В поле **MCP URL** введите конечную точку MCP вашего сервера Генеративный ИИ.
 - Для серверов, привязанных к конкретному проекту: `http://localhost:7860/api/v1/mcp/project/PROJECT_ID/streamable`
 - Для глобального сервера MCP: `http://localhost:7860/api/v1/mcp/streamable`
 - По умолчанию для модуля «Генеративный ИИ» Desktop: `http://localhost:7868/`Все потоки, доступные на целевом сервере, рассматриваются как инструменты.
3. В меню заголовка компонента включите **Режим инструмента**, чтобы использовать компонент с агентом.

4. Подключите порт **Toolset** компонента **Инструменты MCP** к порту **Инструменты** компонента **Агент**.
5. Убедитесь, что к компоненту **Агент** также подключены компоненты **Ввод в чат** и **Вывод чата**.
6. Протестируйте поток, чтобы убедиться, что агент использует ваши потоки для ответов. Откройте **Playground** и введите запрос, задействующий поток из компонента **Инструменты MCP**.
7. Если вы хотите, чтобы агент мог использовать больше инструментов, повторите шаги выше, добавив дополнительные компоненты инструментов с другими серверами или инструментами.

12.2 Параметры компонента «Инструменты MCP»

Название	Тип	Описание
mcp_server	String	Входной параметр. Сервер MCP для подключения. Выберите из ранее настроенных серверов или добавьте новый.
tool	String	Входной параметр. Конкретный инструмент для выполнения на подключённом сервере MCP. Оставьте пустым, чтобы разрешить доступ ко всем инструментам.
use_cache	Boolean	Входной параметр. Включить кэширование сервера MCP и инструментов для повышения производительности. По умолчанию: false.
verify_ssl	Boolean	Входной параметр. Включить проверку SSL-сертификата для HTTPS-соединений. По умолчанию: true.
response	DataFrame	Выходной параметр. Объект DataFrame, содержащий ответ от выполненного инструмента.

12.3 Управление подключёнными серверами MCP

Для управления всеми подключениями к серверам MCP в вашем клиенте Генеративный ИИ нажмите **Серверы MCP** в визуальном редакторе, или нажмите значок профиля, выберите **Настройки** и нажмите **Серверы MCP**.

Чтобы добавить новый сервер MCP, нажмите **Добавить сервер MCP** и следуйте инструкциям в разделе Использование компонента «Инструменты MCP».

Нажмите **Ещё**, чтобы отредактировать или удалить подключение к серверу MCP.

12.4 Изменение переменных окружения сервера MCP через API

Вы можете изменять переменные окружения сервера MCP во время выполнения при запуске потоков через API Генеративный ИИ, настраивая параметры компонента **Инструменты MCP**.

Настройки можно включать в любой запрос API Генеративный ИИ, поддерживающий параметр `tweaks`, — например, в POST-запросы к конечным точкам `/run` или `/webhook`. Для получения дополнительной информации см. Схема ввода (настройки).

Чтобы изменить переменные окружения компонента **Инструменты MCP** через настройки:

1. Откройте поток, содержащий компонент **Инструменты MCP**.
2. Чтобы найти уникальный идентификатор компонента **Инструменты MCP**, нажмите **Управление** в этом компоненте. Идентификатор отображается в панели **Управление**, например `MCPTools-Bzahc`.
3. Отправьте POST-запрос к конечной точке `/run` вашего сервера Генеративный ИИ, включив настройки для компонента **Инструменты MCP**.

Следующие примеры демонстрируют структуру запроса с объектом `env`, вложенным в `mcp_server` в полезной нагрузке `tweaks`:

```
ASTRA_SERVER_ADDRESS = "http://localhost:7860"
FLOW_ID = "your-flow-id"
ASTRA_API_KEY = os.getenv("ASTRA_API_KEY")
MCP_TOOLS_COMPONENT_ID = "MCPTools-Bzahc"

url = f"{ASTRA_SERVER_ADDRESS}/api/v1/run/{FLOW_ID}?stream=false"
headers = {
    "Content-Type": "application/json",
    "x-api-key": ASTRA_API_KEY
}
payload = {
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "What sales data is available to me?",
    "tweaks": {
        MCP_TOOLS_COMPONENT_ID: {
            "mcp_server": {
                "env": {
                    "API_URL": "https://api.example.com",
                    "API_KEY": "your-mcp-server-api-key",
                    "ENVIRONMENT": "production"
                }
            }
        }
    }
}
```

```

}

response = requests.post(url, json=payload, headers=headers)
print(response.json())
const ASTRA_SERVER_ADDRESS = "http://localhost:7860";
const FLOW_ID = "your-flow-id";
const ASTRA_API_KEY = process.env.ASTRA_API_KEY || "";
const MCP_TOOLS_COMPONENT_ID = "MCPTools-Bzahc";

const url = `${ASTRA_SERVER_ADDRESS}/api/v1/run/${FLOW_ID}?stream=false`;

const response = await fetch(url, {
  method: "POST",
  headers: {
    "Content-Type": "application/json",
    "x-api-key": ASTRA_API_KEY,
  },
  body: JSON.stringify({
    output_type: "chat",
    input_type: "chat",
    input_value: "What sales data is available to me?",
    tweaks: {
      [MCP_TOOLS_COMPONENT_ID]: {
        mcp_server: {
          env: {
            API_URL: "https://api.example.com",
            API_KEY: "your-mcp-server-api-key",
            ENVIRONMENT: "production",
          },
        },
      },
    },
  }),
});

const data = await response.json();
console.log(data);
curl --request POST \
  --url "http://SERVER_ADDRESS/api/v1/run/FLOW_ID?stream=false" \
  --header "Content-Type: application/json" \
  --header "x-api-key: ASTRA_API_KEY" \
  --data '{
    "output_type": "chat",
    "input_type": "chat",
    "input_value": "What sales data is available to me?",
    "tweaks": {
      "MCP_TOOLS_COMPONENT_ID": {
        "mcp_server": {
          "env": {
            "API_URL": "https://api.example.com",
            "API_KEY": "your-mcp-server-api-key",
            "ENVIRONMENT": "production"
          }
        }
      }
    }
  }'
```

Замените MCP_TOOLS_COMPONENT_ID, ASTRA_API_KEY, SERVER_ADDRESS и FLOW_ID фактическими значениями из вашего развёртывания Генеративный ИИ.

Генеративный ИИ не определяет автоматически, какие переменные окружения принимает ваш сервер MCP. Чтобы узнать, какие переменные поддерживает сервер, обратитесь к его

документации. Например, сервер Astra DB MCP требует `ASTRA_DB_APPLICATION_TOKEN` и `ASTRA_DB_API_ENDPOINT`, а также поддерживает необязательную переменную `ASTRA_DB_KEYSPACE`.

12.5 Использование глобальных переменных в заголовках сервера MCP

Вы можете использовать глобальные переменные в значениях заголовков сервера MCP для безопасного хранения и передачи API-ключей, токенов аутентификации и других чувствительных значений. Это особенно полезно в сценариях развёртывания, когда необходимо передавать учётные данные пользователя во время выполнения.

Укажите имя глобальной переменной в качестве значения заголовка — Генеративный ИИ автоматически заменит его на фактическое значение перед отправкой запроса к серверу MCP. Генеративный ИИ передаёт только значение токена; самостоятельно он токены не проверяет.

Например, чтобы создать глобальную переменную `TEST_BEARER_TOKEN` для Bearer-аутентификации на сервере MCP:

1. Нажмите значок профиля, выберите **Настройки** и нажмите **Глобальные переменные**, чтобы открыть панель управления глобальными переменными.
2. Создайте глобальную переменную типа **Учётные данные** с именем `TEST_BEARER_TOKEN`.
3. В поле **Уникальный ID** введите значение Bearer-токена вашего сервера MCP. Значение должно включать префикс Bearer с пробелом, например: Bearer eyJhbG....
4. Нажмите **Сохранить переменные**.
5. Нажмите **Серверы MCP** в визуальном редакторе или нажмите значок профиля, выберите **Настройки** и нажмите **Серверы MCP**.
6. Нажмите **Добавить сервер MCP**.
7. Укажите следующие параметры:
 - **Название:** test-mcp-server
 - **Streamable HTTP/SSE URL:** URL вашего сервера MCP, например `http://127.0.0.1:8000/mcp`.
 - **Заголовки:** в поле ключа введите Authorization, в поле значения — `TEST_BEARER_TOKEN` (или точное имя вашей глобальной переменной).
8. Нажмите **Создать сервер**.
Если подключение успешно, Генеративный ИИ отобразит количество инструментов, предоставляемых сервером.
9. Добавьте компонент **Инструменты MCP** в поток.
10. В компоненте **Инструменты MCP** выберите созданный **Сервер MCP**. Конфигурация сервера уже включает настроенные заголовки — дополнительных настроек в компоненте не требуется. Глобальная переменная `TEST_BEARER_TOKEN` автоматически разрешается при каждом запросе к серверу MCP.
11. Опционально: чтобы переопределить или добавить заголовки непосредственно в компоненте **Инструменты MCP**, нажмите на компонент, найдите параметр **Заголовки** в панели инспекции компонента и добавьте пары «ключ — значение». Заголовки, настроенные в компоненте, имеют приоритет над заголовками из конфигурации сервера MCP.

12. Протестируйте поток, чтобы убедиться, что агент использует сервер для ответов. Откройте **Playground** и введите запрос, задействующий инструмент из компонента **Инструменты MCP**.

Генеративный ИИ автоматически подставляет фактическое значение TEST_BEARER_TOKEN перед отправкой запроса. Когда сервер MCP получает запрос, заголовок Authorization уже содержит разрешённое значение токена.

13. MCP сервер

Генеративный ИИ интегрируется с MCP как в роли сервера MCP, так и в роли клиента MCP.

Эта страница описывает использование Генеративный ИИ в роли сервера MCP, который предоставляет ваши потоки в виде инструментов, доступных клиентам MCP при генерации ответов.

Серверы MCP в модуле «Генеративный ИИ» поддерживают транспорт **Streamable HTTP**, а также **Server-Sent Events (SSE)** в качестве резервного варианта. Конфигурация сервера MCP по умолчанию использует транспорт Streamable HTTP по пути /streamable.

Для получения информации об использовании Генеративный ИИ в роли клиента MCP и управлении подключениями к серверам MCP внутри потоков см. Генеративный ИИ в роли клиента MCP.

13.1 Требования

- Проект Генеративный ИИ с как минимум одним потоком, содержащим компонент **Вывод чата**. Компонент **Вывод чата** необходим для использования потока в качестве инструмента MCP.
- Любая LTS-версия Node.js — если вы хотите использовать MCP Inspector для тестирования и отладки потоков.
- Установленный ngrok и ngrok authToken — если вы хотите развернуть публичный сервер Генеративный ИИ.

13.2 Публикация потоков как инструментов MCP

При создании проектов Генеративный ИИ система автоматически добавляет проект в конфигурацию сервера MCP и делает потоки проекта доступными как инструменты MCP.

Если на вашем сервере Генеративный ИИ включена аутентификация (AUTO_LOGIN=false), сервер MCP проекта автоматически настраивается с аутентификацией по API-ключу — генерируется новый ключ специально для доступа к потокам этого проекта. Для получения дополнительной информации см. Аутентификация сервера MCP.

Отключение автоматической конфигурации сервера MCP

Чтобы отключить автоматическую конфигурацию сервера MCP для новых проектов, установите переменную окружения `ASTRA_ADD_PROJECTS_TO_MCP_SERVERS` в значение `false`. Для получения дополнительной информации см. Переменные окружения сервера MCP.

Выборочное включение и отключение серверов MCP

Независимо от того, включена ли автоматическая конфигурация, вы можете выборочно включать и отключать проекты, доступные как инструменты MCP:

1. На странице **Проекты** нажмите вкладку **Сервер MCP**, или при редактировании потока нажмите **Поделиться** и выберите **Сервер MCP**.
Раздел **Потоки / Инструменты** отображает потоки, которые в данный момент предоставляются как инструменты этого сервера MCP.
2. Нажмите **Редактировать инструменты** и выберите потоки, которые хотите сделать доступными. Чтобы исключить поток, снимите галочку в первом столбце.

3. Закройте диалоговое окно **Инструменты сервера MCP**, чтобы сохранить изменения.

Редактирование названий и описаний инструментов

Названия и описания инструментов помогают клиентам MCP определять, какие действия предоставляют ваши потоки и когда их использовать. Рекомендуется задавать чёткие и информативные названия и описания для всех инструментов.

1. На странице **Проекты** нажмите вкладку **Сервер MCP**, или при редактировании потока нажмите **Поделиться** и выберите **Сервер MCP**.
2. Нажмите **Редактировать инструменты**.
3. Нажмите на поле **Описание** или **Инструмент**, которое хотите изменить:
 - **Название инструмента** — введите название, которое чётко указывает на то, что делает поток при использовании агентом.
 - **Описание инструмента** — введите описание, точно и полно отражающее конкретные действия, выполняемые потоком.
4. Закройте диалоговое окно **Инструменты сервера MCP**, чтобы сохранить изменения.

Важность названий и описаний инструментов

Клиенты MCP используют названия и описания для определения, какие инструменты применять при генерации ответов.

Поскольку клиенты MCP рассматривают ваш проект Генеративный ИИ как единый сервер MCP со всеми включёнными потоками в виде инструментов, неясные названия и описания могут привести к неправильному или непоследовательному выбору инструментов.

Название инструмента по умолчанию — это идентификатор потока, например adbbf8c7-0a34-493b-90ea-5e8b42f78b66, который не содержит никакой информации о назначении потока.

Считайте названия и описания инструментов аналогами имён функций и комментариев в коде. Используйте чёткие формулировки, описывающие задачи, которые решают ваши потоки.

Пример: название и описание инструмента

Предположим, вы создаёте поток на основе шаблона **Document Q&A** для анализа резюме и задаёте ему следующие название и описание:

- **Название инструмента:** document_qa_for_resume
- **Описание инструмента:** A flow for analyzing Emily's resume.

После подключения вашего сервера MCP Генеративный ИИ к Cursor вы можете задать вопрос о резюме, например What job experience does Emily have?. Используя контекст из названия и описания инструмента, агент выбирает инструмент document_qa_for_resume для формирования ответа. При необходимости агент запрашивает разрешение перед использованием инструмента.

Если вы спросите о другом резюме, например What job experience does Alex have?, агент может решить, что document_qa_for_resume неприменим к этому запросу, поскольку описание указывает на резюме Эмили. В таком случае агент использует другой доступный инструмент или сообщает об отсутствии доступа к нужным данным.

13.3 Подключение клиентов к серверу MCP

Генеративный ИИ предоставляет автоматическую установку и фрагменты кода для подключения серверов MCP к локальным клиентам.

Вариант JSON позволяет подключить сервер MCP Генеративный ИИ к любому локальному или удалённому клиенту MCP. Процесс аналогичен для любого совместимого с MCP клиента.

1. Установите любой совместимый с MCP клиент.

В этих шагах в качестве примера используется Cursor, однако процесс в целом одинаков для всех клиентов с небольшими различиями в деталях, специфичных для каждого клиента.

2. В клиенте добавьте новый сервер MCP через пользовательский интерфейс или файл конфигурации.

Например, в Cursor перейдите в **Настройки Cursor** → **MCP** → **Добавить новый глобальный сервер MCP**, чтобы открыть глобальный файл конфигурации mcp.json.

3. Рекомендуется: настройте аутентификацию для вашего сервера MCP.

4. В Генеративный ИИ на странице **Проекты** нажмите вкладку **Сервер MCP**.

5. Нажмите вкладку **JSON**, скопируйте фрагмент кода для вашей операционной системы и вставьте его в файл конфигурации MCP вашего клиента. Например:

```
{
  "mcpServers": {
    "PROJECT_NAME": {
      "command": "uvx",
      "args": [
        "mcp-proxy",
        "--transport",
        "streamablehttp",
        "http://SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable"
      ]
    }
  }
}
```

Вкладка **Сервер MCP** автоматически подставляет значения `SERVER_ADDRESS` и `PROJECT_ID`.

Адрес сервера по умолчанию — `http://localhost:7860`. При использовании публичного сервера Генеративный ИИ адрес подставляется автоматически.

Если ваш сервер требует аутентификации, добавьте API-ключ или настройки OAuth в конфигурацию. Подробнее см. Аутентификация сервера MCP.

6. Чтобы добавить переменные окружения в команду сервера MCP, добавьте объект `env` с парами «ключ — значение». Например:

```
{
  "mcpServers": {
    "PROJECT_NAME": {
      "command": "uvx",
      "args": [
        "mcp-proxy",
        "--transport",
        "streamablehttp",
        "http://SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable"
      ],
      "env": {
        "KEY": "VALUE"
      }
    }
  }
}
```

Не помещайте API-ключи в объект `env` — эти переменные предназначены исключительно для процесса `mcp-proxy`. API-ключи добавляйте в `args`. Пример см. в разделе Аутентификация сервера MCP.

7. Сохраните и закройте файл конфигурации MCP вашего клиента.
8. Убедитесь, что ваш сервер MCP Генеративный ИИ отображается в списке серверов MCP клиента. При необходимости перезапустите клиент, чтобы применить изменения конфигурации.

Вариант автоустановки доступен только для определённых клиентов MCP. Автоустановка требует локальной установки клиента, чтобы Генеративный ИИ мог записать файл конфигурации. Если ваш клиент не поддерживается, установлен удалённо или требуется передать дополнительные переменные окружения, используйте вариант **JSON**. 1. Установите Cursor, Claude или Windsurf на тот же компьютер, где работает ваш сервер Генеративный ИИ.

2. Рекомендуется: настройте аутентификацию для вашего сервера MCP.
3. В Генеративный ИИ на странице **Проекты** нажмите вкладку **Сервер MCP**.
4. На вкладке **Автоустановка** найдите вашего клиента MCP и нажмите **Добавить**.

Сервер MCP вашего проекта автоматически добавляется в файл конфигурации локального клиента. Например, для Cursor конфигурация записывается в файл `mcp.json`.

Генеративный ИИ пытается добавить конфигурацию даже если выбранный клиент не установлен. Чтобы проверить установку, просмотрите список доступных серверов MCP в вашем клиенте.

После подключения клиента MCP к серверу MCP вашего проекта Генеративный ИИ ваши потоки регистрируются как инструменты. Клиент определяет, когда использовать инструменты, исходя из ваших запросов, и при необходимости запрашивает разрешения. Для получения дополнительной информации см. документацию MCP для вашего клиента, например документацию MCP Cursor.

13.4 Аутентификация сервера MCP

Каждый проект Генеративный ИИ имеет собственный сервер MCP с отдельными настройками аутентификации.

При создании нового проекта Генеративный ИИ автоматически настраивает аутентификацию сервера MCP в соответствии с настройками вашего сервера. Если аутентификация включена (`AUTO_LOGIN=false`), автоматически генерируется API-ключ для доступа к потокам проекта.

Чтобы настроить аутентификацию сервера MCP, перейдите на страницу **Проекты**, нажмите вкладку **Сервер MCP**, выберите **Редактировать аутентификацию** и выберите предпочитаемый метод:

При аутентификации через API-ключ Генеративный ИИ фрагменты кода **JSON** и конфигурация **Автоустановки** автоматически включают аргументы `--headers` и `x-api-key` в массив `args`.

Нажмите **Сгенерировать API-ключ**, чтобы автоматически вставить новый ключ в шаблон кода. Или замените `YOUR_API_KEY` существующим ключом API Генеративный ИИ.

Добавьте API-ключ в конфигурацию через три отдельные записи в `args`:

```
{  
  "mcpServers": {  
    "PROJECT_NAME": {
```

```

"command": "uvx",
"args": [
  "mcp-proxy",
  "--transport",
  "streamablehttp",
  "--headers",
  "x-api-key",
  "YOUR_API_KEY",
  "http://SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable"
]
}
}
}

```

При включении OAuth Генеративный ИИ автоматически запускает экземпляр MCP Composer для вашего проекта, создавая безопасный клиентский прокси между клиентами MCP и mcp-proxy.

Интеграция OAuth позволяет серверу MCP аутентифицировать пользователей и приложения через любой совместимый с OAuth 2.0 сервис. При подключении пользователи перенаправляются к выбранному провайдеру OAuth для аутентификации, после чего получают доступ к потокам как инструментам MCP.

Перед настройкой OAuth создайте OAuth-приложение у внешнего провайдера OAuth 2.0, зарегистрируйте сервер Генеративный ИИ в качестве OAuth-клиента и получите необходимые значения.

В следующей таблице описаны обязательные поля. В качестве примера используется GitHub OAuth. Используйте реальные значения из вашего развёртывания.

Поле	Описание	Источник	Пример
Хост	Хост сервера OAuth	Значение по умолчанию MCP Composer	localhost
Порт	Порт сервера OAuth	Значение по умолчанию MCP Composer	9000
URL сервера	Полный URL сервера OAuth	Объединяет хост и порт	http://localhost:9000
URL обратного вызова	URL обратного вызова OAuth	Задаётся при регистрации приложения; должен точно совпадать со значением у провайдера	http://localhost:9000/auth/idaas/callback
Идентификатор клиента	Идентификатор OAuth-клиента	От вашего провайдера OAuth	Ov23li9vx2grVL61qjb
Секрет клиента	Секрет OAuth-клиента	От вашего провайдера OAuth	1234567890abcdef12345678
Authorization URL	Точка авторизации OAuth	От вашего провайдера OAuth	https://github.com/login/oauth/authorize

Поле	Описание	Источник	Пример
Token URL	Точка получения токенов OAuth	От вашего провайдера OAuth	<code>https://github.com/login/oauth/access_token</code>
MCP Scope	Область для операций MCP	Задаётся вами; начиная с версии 1.6 доступно только user	<code>user</code>
Provider Scope	Область провайдера OAuth	Задаётся вами; начиная с версии 1.6 доступно только openid	<code>openid</code>

Настройка аутентификации OAuth:

1. Выберите **OAuth** в качестве типа аутентификации.
2. Заполните параметры OAuth значениями из вашего развёртывания. Все поля обязательны.
Учётные данные OAuth шифруются и безопасно хранятся в базе данных Генеративный ИИ.
3. Нажмите **Сохранить**.
Фрагменты кода **JSON** и конфигурация **Автоустановки** автоматически обновляются значениями OAuth для новых подключений. Существующие установки необходимо обновить вручную.
4. Если сервер MCP Генеративный ИИ уже установлен в клиент MCP, обновите конфигурацию клиента. Метод зависит от способа первоначальной установки:
 - **Автоустановка** — вручную обновите файл конфигурации клиента, используя обновлённый JSON-фрагмент с вкладки **JSON**, или повторите шаги автоустановки.
 - **Вариант JSON** — скопируйте обновлённый фрагмент с вкладки **JSON** и замените им текущую конфигурацию.
 - **Новые подключения** — используйте вариант **Автоустановка** или **JSON**; настройки OAuth включаются автоматически.

После включения OAuth и обновления конфигурации клиента при каждой попытке аутентификации открывается окно обратного вызова OAuth. Успешная аутентификация возвращает сообщение `Authentication complete. You may close this window`. Если клиент не открывает окно OAuth, попробуйте перезапустить его для применения обновлённой конфигурации.

Если аутентификация не настроена, сервер MCP становится публичной точкой доступа — любой может подключиться без учётных данных. Используйте эту опцию только в доверенной среде.

13.5 Переменные окружения сервера MCP

Следующие переменные окружения управляют поведением серверов MCP ваших проектов:

Переменная	Тип	По умолчанию	Описание
ASTRA_MCP_SERVER_ENABLE	Boolean	true	Определяет, следует ли инициализировать сервер MCP для каждого проекта. Если false, серверы MCP не инициализируются.
ASTRA_MCP_SERVER_ENABLE_PROGRESS_NOTIFICATIONS	Boolean	false	Если true, серверы MCP отправляют уведомления о прогрессе выполнения.
ASTRA_MCP_SERVER_TIMEOUT	Integer	20	Количество секунд ожидания до завершения операции сервера MCP из-за плохого соединения или длительного запроса.
ASTRA_MCP_MAX_SESSIONS_PER_SERVER	Integer	10	Максимальное количество MCP-сессий для каждого уникального сервера.
ASTRA_ADD_PROJECTS_TO_MCP_SERVERS	Boolean	true	Определяет, следует ли автоматически добавлять новые проекты в конфигурацию серверов MCP. Если false, проекты добавляются вручную.

Внешнее развёртывание сервера MCP

Для внешнего развёртывания сервера MCP Генеративный ИИ см. Развёртывание публичного сервера Генеративный ИИ.

13.6 Тестирование и отладка с помощью MCP Inspector

note Требования к Node.js MCP Inspector требует любую LTS-версию Node.js. MCP Inspector — популярный инструмент для тестирования и отладки серверов MCP. Используйте его для мониторинга потоков и наблюдения за тем, как они потребляются сервером MCP.

1. Установите MCP Inspector:

```
npx @modelcontextprotocol/inspector
```

Для получения дополнительной информации о настройке, включая указание порта прокси, см. MCP Inspector на GitHub.

2. Откройте браузер и перейдите к интерфейсу MCP Inspector. Адрес по умолчанию: `http://localhost:6274`.
3. Введите параметры подключения к серверу MCP вашего проекта. Значения зависят от настроенного метода аутентификации.

- **Тип транспорта:** выберите **STDIO**.
- **Команда:** `uvx`
- **Аргументы:** введите список аргументов через пробелы, заменив `YOUR_API_KEY`, `ASTRA_SERVER_ADDRESS` и `PROJECT_ID` реальными значениями:

```
mcp-proxy --headers x-api-key YOUR_API_KEY  
http://ASTRA_SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable
```

- **Тип транспорта:** выберите **STDIO**.
- **Команда:** `uvx`
- **Аргументы:** введите список аргументов через пробелы, заменив `OAUTH_SERVER_URL` URL вашего сервера OAuth:

```
mcp-composer --mode stdio --sse-url http://localhost:9000/sse --disable-composer-tools --client_auth_type oauth
```

- **Тип транспорта:** выберите **SSE**.
- **URL:** введите конечную точку сервера MCP Генеративный ИИ. Например:

```
http://localhost:7860/api/v1/mcp/project/d359cbd4-6fa2-4002-9d53-fa05c645319c/streamable
```

4. Нажмите **Подключиться**.

При успешном подключении во вкладке **Инструменты** отображаются потоки вашего проекта. Здесь можно наблюдать, как MCP регистрирует потоки как инструменты, и тестировать инструменты с пользовательскими входными данными.

5. Чтобы выйти из MCP Inspector, нажмите `Ctrl+C` в терминальном окне, где он был запущен.

13.7 Устранение неполадок

Рекомендации по устранению неполадок серверов и клиентов MCP см. в разделе Устранение неполадок Генеративный ИИ: проблемы MCP.

14. Astra DB MCP

Это руководство демонстрирует использование Генеративный ИИ в роли клиента MCP с помощью компонента **Инструменты MCP** для подключения сервера Astra DB MCP от DataStax к агентному потоку.

1. Установите LTS-версию Node.js.

uvx включён в пакет uv, поставляемый вместе с модулем «Генеративный ИИ».

Для использования команд сервера на основе prx сначала установите LTS-версию Node.js. Если вы запускаете Генеративный ИИ в Docker, установите Node.js внутри образа контейнера и пересоберите его, чтобы серверы MCP на основе prx были доступны во время выполнения. Для получения дополнительной информации см. Управление пакетами.

Пример сервера MCP на основе prx в модуле «Генеративный ИИ» см. в разделе Подключение сервера Astra DB MCP к модулю «Генеративный ИИ». 2. Создайте API-ключ OpenAI.

3. Создайте базу данных Astra DB Serverless (vector), если у вас её ещё нет.
4. Получите конечную точку API Astra DB и токен приложения Astra с ролью **Администратор баз данных**. Для получения дополнительной информации см. Создание токена базы данных.
5. Создайте поток на основе шаблона **Простой агент**, чтобы следовать этому руководству.

Вы также можете использовать существующий поток или создать пустой поток.

6. Удалите инструмент **URL** и замените его компонентом **Инструменты MCP**.
7. Настройте компонент **Инструменты MCP** следующим образом:
 1. В поле **Сервер MCP** нажмите **Добавить сервер MCP**.
 2. Выберите режим **Stdio**.
 3. В поле **Название** введите имя для сервера MCP.
 4. В поле **Команда** введите следующую команду для подключения к серверу Astra DB MCP:

```
prx -u @datastax/astra-db-mcp
```

5. В полях **Переменные окружения** добавьте переменные **ASTRA_DB_APPLICATION_TOKEN** и **ASTRA_DB_API_ENDPOINT** со значениями из вашей базы данных Astra.

Переменные окружения, объявленные в файле `.env` Генеративный ИИ, можно использовать в командах сервера MCP. Однако ссылаться на глобальные переменные, объявленные в интерфейсе Генеративный ИИ, нельзя.

Если вы хотите использовать переменные **ASTRA_DB_APPLICATION_TOKEN** и **ASTRA_DB_API_ENDPOINT**, добавьте их в файл `.env` Генеративный ИИ и перезапустите сервер. Для получения дополнительной информации см. Глобальные переменные. Добавляйте каждую переменную отдельно. Чтобы добавить ещё одно поле переменной, нажмите **Добавить ещё**.

```
ASTRA_DB_APPLICATION_TOKEN=AstraCS:...
```

```
ASTRA_DB_API_ENDPOINT=https://...-us-east-2.apps.astra.datastax.com
```

8. В компоненте **Агент** добавьте ваш API-ключ OpenAI.

По умолчанию используется модель OpenAI. Чтобы выбрать другую модель, измените поля **Провайдер модели**, **Название модели** и **Ключ API** соответственно.
9. Откройте **Playground** и задайте агенту запрос What collections are available?

Поскольку Генеративный ИИ подключён к вашей базе данных Astra DB через сервер MCR, агент выбирает нужный инструмент и обращается к базе данных для получения ответа.

15. Ключи API и аутентификация

Никогда не открывайте порты Генеративный ИИ напрямую в Интернет без надлежащих мер безопасности. Установите `ASTRA_AUTO_LOGIN=False`, используйте нестандартный `ASTRA_SECRET_KEY` и разместите ваш сервер Генеративный ИИ за обратным прокси с включённой аутентификацией. Для получения дополнительной информации см. Запуск сервера Генеративный ИИ с включённой аутентификацией. Учётные данные аутентификации помогают предотвратить несанкционированный доступ к вашему серверу Генеративный ИИ, потокам и сервисам, подключённым через компоненты.

Существует три типа учётных данных, которые вы используете в модуле «Генеративный ИИ»:

- **Генеративный ИИ API keys:** Для аутентификации с API Генеративный ИИ и авторизации серверных действий Генеративный ИИ, таких как запуск потоков и загрузка файлов.
- **Component API keys:** Для аутентификации между Генеративный ИИ и сервисом, подключённым через компонент, например, поставщиком модели или сторонним API.
- **Authentication environment Переменная s:** Эти переменные окружения определяют, как Генеративный ИИ обрабатывает аутентификацию и авторизацию пользователей.

15.1 Ключи API Генеративный ИИ

Вы можете использовать ключи API Генеративный ИИ для программного взаимодействия с модулем «Генеративный ИИ».

По умолчанию большинство конечных точек API Генеративный ИИ, таких как `/v1/run/$FLOW_ID`, требуют аутентификации с помощью ключа API Генеративный ИИ.

Генеративный ИИ проверяет ключи API по ключам, хранящимся в базе данных, но вы можете настроить Генеративный ИИ для проверки ключей API с помощью переменной окружения. Для получения дополнительной информации см. `ASTRA_API_KEY_SOURCE`.

Чтобы требовать аутентификацию ключом API для конечных точек вебхуков потока, используйте переменную окружения `ASTRA_WEBHOOK_AUTH_ENABLE`. Для настройки аутентификации серверов Генеративный ИИ MCP см. `Use Генеративный ИИ as an MCP server`.

Права доступа ключа API Генеративный ИИ

Ключ API Генеративный ИИ наследует привилегии пользователя, который его создал. Это означает, что созданные вами ключи API имеют те же разрешения и доступ, что и у вас, включая доступ к вашим потокам, компонентам и базе данных Генеративный ИИ. Ключ API Генеративный ИИ нельзя использовать для доступа к ресурсам за пределами вашего собственного сервера Генеративный ИИ.

В однопользовательских средах вы всегда являетесь суперпользователем, и ваши ключи API Генеративный ИИ всегда имеют привилегии суперпользователя.

В многопользовательских средах пользователи, не являющиеся суперпользователями, не могут использовать свои ключи API для доступа к ресурсам других пользователей. Суперпользователи могут запускать только свои собственные потоки и не могут запускать потоки, принадлежащие другим пользователям. Вы должны запустить ваш сервер Генеративный ИИ с включённой аутентификацией, чтобы позволить суперпользователям управлять пользователями и создавать аккаунты без прав суперпользователя.

Создание ключа API Генеративный ИИ

Вы можете сгенерировать ключ API Генеративный ИИ в **Настройках** Генеративный ИИ или с помощью Генеративный ИИ CLI.

Опция CLI требуется, если ваш сервер Генеративный ИИ работает в режиме `--backend-only`.

1. В заголовке Генеративный ИИ нажмите на значок профиля, затем выберите **Настройки**.
2. Нажмите **Генеративный ИИ API Ключи**, затем нажмите **Добавить Новый**.
3. Назовите ваш ключ, а затем нажмите **Создать Ключ API**.
4. Скопируйте ключ API и храните его в безопасном месте.

Если вы обслуживаете ваш поток с помощью `--backend-only=true`, вы не можете создавать ключи API в **Настройках** Генеративный ИИ, поскольку фронтенд не запущен. В этом случае вам необходимо создавать ключи API с помощью Генеративный ИИ CLI.

1. Рекомендуется: Запустить ваш сервер Генеративный ИИ с включённой аутентификацией.

Команда Генеративный ИИ рекомендует включать аутентификацию по соображениям безопасности, чтобы предотвратить неавторизованное создание ключей API и суперпользователей, особенно в производственных средах. Если аутентификация не включена (`ASTRA_AUTO_LOGIN=True`), все пользователи фактически являются суперпользователями и могут создавать ключи API с помощью Генеративный ИИ CLI.

2. Создайте ключ API с помощью Генеративный ИИ `api-key`:

```
uv run astra api-key
```

Все ключи API, созданные с помощью Генеративный ИИ CLI, имеют привилегии суперпользователя, поскольку команда требует аутентификации суперпользователя, а ключи API Генеративный ИИ наследуют привилегии пользователя, который их создал.

Использование ключа API Генеративный ИИ

Для аутентификации запросов к API Генеративный ИИ передайте ваш ключ API Генеративный ИИ в заголовке `x-api-key` или параметре запроса.

```
curl -X POST \
  "http://$SERVER_ADDRESS/api/v1/run/$FLOW_ID?stream=false" \
  -H "Content-Type: application/json" \
  -H "x-api-key: $ASTRA_API_KEY" \
  -d '{"inputs": {"text":""}, "tweaks": {}}'
curl -X POST \
  "http://$SERVER_ADDRESS/api/v1/run/$FLOW_ID?x-api-key=$API_KEY" \
  -H "Content-Type: application/json" \
  -d '{"inputs": {"text":""}, "tweaks": {}}'
```

Для получения дополнительной информации о формировании запросов к API Генеративный ИИ см. [Get started with the Генеративный ИИ API](#) и [Trigger flows with the Генеративный ИИ API](#).

Отслеживание использования ключа API

По умолчанию Генеративный ИИ отслеживает использование ключей API через записи `total_uses` и `last_used_at` в вашем Генеративный ИИ базе данных.

Чтобы отключить отслеживание ключей API, установите `ASTRA_DISABLE_TRACK_APIKEY_USAGE=True` в ваших переменных окружения Генеративный ИИ. Это может помочь избежать конфликтов базы данных в периоды высокой конкуренции.

Отзыв ключа API

Чтобы отозвать и удалить ключ API, выполните следующие действия:

1. В заголовке Генеративный ИИ нажмите на значок вашего профиля, затем выберите **Настройки**.
2. Нажмите **Генеративный ИИ API Ключи**.
3. Выберите ключи, которые хотите удалить, и нажмите **Удалить**.

Это действие немедленно делает ключ недействительным и предотвращает его дальнейшее использование.

15.2 Ключи API компонентов

Ключи API компонентов предоставляют доступ к внешним сервисам, которые вызываются компонентами в ваших потоках, таким как поставщики моделей, базы данных или сторонние API. Это не ключи API Генеративный ИИ и не общие учетные данные приложения.

В Генеративный ИИ вы можете хранить ключи API компонентов в глобальных переменных в разделе **Settings** или импортировать их из среды выполнения. Для получения дополнительной информации см. Глобальные переменные.

Вы создаёте и управляете ключами API компонентов на платформе поставщика услуг. Генеративный ИИ хранит только зашифрованное значение ключа или безопасную ссылку на ключ, хранящийся в другом месте; он не управляет фактическими учётными данными у источника. Это означает, что удаление глобальной переменной из модуля «Генеративный ИИ» не удаляет и не делает недействительным реальный ключ API в системе поставщика услуг. Вы должны удалить или обновить ключи API компонентов непосредственно через интерфейс или API поставщика услуг.

Для повышения безопасности вы можете установить `Генеративный ИИ_REMOVE_API_KEYS=True`, чтобы исключить ключи API и токены из данных потока в вашем Генеративный ИИ базе данных. Кроме того, при экспорте потоков вы можете выбрать исключение ключей API из экспортированного JSON потока.

16. Глобальные переменные

Используйте глобальные переменные для хранения и повторного использования учётных данных и общих значений во всех своих потоках. Глобальные переменные используются компонентами в потоке — их можно указать в любом поле, обозначенном иконкой глобальной переменной.

В отличие от переменных окружения, таких как `ASTRA_PORT` или `ASTRA_LOG_LEVEL`, которые предназначены для более широких настроек работы Генеративный ИИ, глобальные переменные задаются на уровне потоков. Тем не менее Генеративный ИИ также может получать глобальные переменные из переменных окружения.

Генеративный ИИ хранит глобальные переменные во внутренней базе данных и шифрует их значения с помощью секретного ключа.

16.1 Создание глобальной переменной

Чтобы создать новую глобальную переменную, выполните следующие шаги.

1. В заголовке Генеративный ИИ нажмите значок профиля и выберите **Настройки**.
2. Нажмите **Глобальные переменные**.
3. Нажмите **Добавить новую**.
4. В диалоговом окне **Создать переменную** введите имя переменной в поле **Название переменной**.
5. Опционально: выберите **Тип** для глобальной переменной. Доступные типы: **Универсальный** (по умолчанию) и **Полномочия**.
Генеративный ИИ шифрует глобальные переменные обоих типов. Однако переменные типа **Универсальный** не маскируются в визуальном редакторе, тогда как переменные типа **Полномочия** маскируются. Поля **Идентификатор сеанса** не принимают переменные типа **Полномочия** (маскированные).
6. Введите **Уникальный ID** для глобальной переменной.
7. Опционально: используйте меню **Применить к полям**, чтобы выбрать одно или несколько полей, к которым Генеративный ИИ будет автоматически применять переменную. Например, если выбрать **API-ключ OpenAI**, Генеративный ИИ автоматически подставит переменную в любое поле **API-ключ OpenAI**.
8. Нажмите **Сохранить переменную**.

Теперь вы можете выбрать глобальную переменную в любом текстовом поле ввода, обозначенном иконкой **Шар**.

16.2 Редактирование глобальной переменной

1. В заголовке Генеративный ИИ нажмите значок профиля и выберите **Настройки**.
2. Нажмите **Глобальные переменные**.
3. Нажмите на глобальную переменную, которую хотите отредактировать.
4. В диалоговом окне **Обновить переменную** вы можете изменить следующие поля: **Название переменной**, **Уникальный ID** и **Применить к полям**.
5. Нажмите **Обновить переменную**.

16.3 Удаление глобальной переменной

Удаление глобальной переменной навсегда удаляет её значение из базы данных. Потоки, ссылающиеся на удалённую переменную, будут завершаться с ошибкой.

1. В заголовке Генеративный ИИ нажмите значок профиля и выберите **Настройки**.
2. Нажмите **Глобальные переменные**.
3. Установите флажок рядом с глобальной переменной, которую хотите удалить.
4. Нажмите **Удалить**.

Глобальная переменная удалена из базы данных.

17. Хранилище и память

17.1 Управление файлами

Каждый сервер Генеративный ИИ имеет систему управления файлами, где можно хранить файлы для использования в потоках.

Файлы, загруженные в систему управления файлами Генеративный ИИ, хранятся в бэкенде хранилища (локальном или AWS S3) и доступны всем вашим потокам.

Централизованное хранение файлов позволяет повторно использовать их в разных потоках без повторной загрузки.

Интерфейс управления файлами

С помощью интерфейса управления файлами вы можете загружать файлы с локального компьютера на сервер Генеративный ИИ и управлять всеми загруженными файлами.

1. Перейдите к системе управления файлами Генеративный ИИ:
 - **Генеративный ИИ Desktop**: на странице **Проекты** нажмите **Мои файлы** под списком проектов.
 - **Генеративный ИИ OSS**: в браузере перейдите по адресу `/files` вашего сервера, например `http://localhost:7860/files`. При необходимости измените базовый URL.
 - **Только бэкенд**: для программного управления файлами используйте эндпоинты файлов API Генеративный ИИ. Последующие шаги предполагают использование интерфейса управления файлами.
2. На странице **Мои файлы** нажмите **Загрузить**.
3. Выберите один или несколько файлов для загрузки.

После загрузки файлы можно переименовывать, скачивать, копировать и удалять. Чтобы удалить файл, наведите курсор на его значок, выберите его и нажмите **Удалить** — можно удалить несколько файлов за один раз. Чтобы скачать файл, наведите курсор на его значок, выберите его и нажмите **Скачать** — при скачивании нескольких файлов они упаковываются в zip-архив.

Загрузка файлов и управление ими через API

С помощью API Генеративный ИИ вы можете загружать файлы в систему управления файлами, управлять ими, а также передавать файлы в потоки программно во время выполнения.

Подробнее и с примерами см. в разделах **Эндпоинты файлов** и **Создание чат-бота с поддержкой файлов**.

Использование файлов в потоке

Чтобы использовать файлы из системы управления файлами в потоке, добавьте компонент, принимающий файловый ввод, — например компонент **Чтение файла**.

Добавьте компонент **Чтение файла** в поток, нажмите **Выбрать файлы** и выберите нужные файлы из списка **Мои файлы**.

Список содержит все файлы на сервере, однако для выбора доступны только типы файлов, поддерживаемые компонентом **Чтение файла**. Для работы с другим типом файла используйте подходящий компонент или конвертируйте файл в поддерживаемый формат перед загрузкой.

Подробнее см. в разделе **Компонент Чтение файла**.

Загрузка файлов во время выполнения

Вы можете использовать заранее загруженные файлы, а также передавать файлы во время выполнения, если поток принимает файловый ввод. Чтобы включить файловый ввод в потоке:

1. Добавьте компонент **Чтение файла** в поток.
2. Нажмите **Поделиться**, выберите **API-доступ**, затем нажмите **Схема ввода**, чтобы добавить tweaks в тело запроса в автоматически сгенерированных фрагментах кода.
3. Разверните раздел **Файл**, найдите строку **Файлы** и включите **Разрешить ввод**, чтобы параметр можно было задавать во время выполнения через API.
4. Закройте панель **Схема ввода**, чтобы вернуться к панели **API-доступ**. Тело запроса в каждом фрагменте кода теперь включает tweaks с ID компонента **Чтение файла** и ключом path:

```
"tweaks": {  
  "File-qYD5w": {  
    "path": []  
  }  
}
```

5. При программном запуске потока скрипт должен сначала загрузить файл в систему управления файлами Генеративный ИИ, а затем передать полученный file_path в параметр path запроса /run:

```
"tweaks": {  
  "FILE_COMPONENT_ID": {  
    "path": [ "file_path" ]  
  }  
}
```

Полный пример см. в разделах Создание чат-бота с поддержкой файлов и Эндпоинты файлов.

Чтобы передать несколько файлов, укажите несколько значений file_path в массиве path, например ["path1", "path2"].

Загрузка изображений

Генеративный ИИ поддерживает изображения в формате base64 следующих типов:

- PNG
- JPG/JPEG
- GIF
- BMP
- WebP

Изображения можно загружать в чат-интерфейс **Playground** или передавать через API Генеративный ИИ во время выполнения.

- В **Playground** перетащите изображение в область ввода или нажмите значок **Прикрепить изображение**, чтобы выбрать файл.
- При вызове потока через эндпоинт /api/v1/run/\$FLOW_ID используйте параметр files для передачи изображения в виде строки base64:

```
curl -X POST "http://$LANGFLOW_SERVER_ADDRESS/api/v1/run/$FLOW_ID" \  
-H "Content-Type: application/json" \  
-H "x-api-key: $ASTRA_API_KEY" \  
-d '{  
  "session_id": "custom_session_123",  
  "input_value": "What is in this image?",  
  "input_type": "chat",  
  "output_type": "chat",  
  "files": ["data:image/png;base64,iVBORw0KGgoAAAANSUhEUgAA..."]  
}'
```


Для специализированной обработки изображений просмотрите **Пакеты** или создайте собственные компоненты.

Работа с видеофайлами

Для работы с видео используйте компоненты-пакеты **Twelve Labs** и **YouTube** из раздела **Пакеты**.

17.2 Управление памятью

Генеративный ИИ предоставляет гибкие варианты управления памятью для хранения и извлечения данных, относящихся к вашим потокам и серверу Генеративный ИИ. Это включает основные таблицы базы данных, управление файлами и кэширование, а также память чата.

Хранение памяти чата

Потоки с компонентами **Ввод в чат** или **Вывод чата** создают историю чата, которая сохраняется в таблице `messages` базы данных Генеративный ИИ. Минимально это служит журналом чата, однако функционально это не то же самое, что память чата, предоставляющая исторический контекст для языковой модели.

Для хранения и извлечения памяти чата в потоках используйте компонент **История сообщений** или встроенную память чата компонента **Агент**.

Как работает память чата?

Память чата — это кэш для языковой модели или агента, позволяющий сохранять прошлые разговоры и использовать этот контекст в будущих взаимодействиях. Например, если пользователь уже сообщил языковой модели своё имя, модель может получить эту информацию из памяти чата, не запрашивая её повторно.

Память чата отличается от памяти векторного хранилища: она специально предназначена для хранения и извлечения сообщений чата из баз данных, а не для семантического поиска по фрагментам текста.

Компоненты, поддерживающие память чата (например, **Агент** и **История сообщений**), предоставляют доступ к своим базам данных в качестве памяти. Этот механизм специально оптимизирован для восстановления контекста из прошлых разговоров.

Идентификатор сеанса и память чата

История чата и память группируются по идентификатору сеанса (`session_id`).

По умолчанию идентификатор сеанса совпадает с идентификатором потока: все сообщения чата для потока сохраняются под одним идентификатором как единая сессия.

Для лучшего разделения памяти чата, особенно в потоках с несколькими пользователями, рекомендуется использовать пользовательские идентификаторы сеансов. Например, использование идентификаторов пользователей в качестве идентификаторов сеансов позволяет хранить историю чата каждого пользователя отдельно, изолируя контекст их бесед от чатов других пользователей.

Варианты памяти чата

Место и способ хранения памяти чата зависят от компонентов, используемых в потоке:

- **Компонент Агент** — имеет встроенную память чата, включённую по умолчанию. Она позволяет агенту извлекать и использовать сообщения из предыдущих разговоров с тем же идентификатором сеанса. Все сообщения сохраняются в хранилище Генеративный ИИ; компонент предоставляет минимальные параметры настройки памяти, такие как количество извлекаемых сообщений.

Встроенной памяти чата компонента **Агент** достаточно для большинства сценариев.

Если требуется внешнее хранилище памяти чата, извлечение памяти вне контекста чата или использование памяти чата с компонентом языковой модели (не агентом) — используйте компонент **История сообщений**.

- **Компонент История сообщений** — по умолчанию сохраняет и извлекает память из хранилища Генеративный ИИ, если не подключён сторонний компонент памяти чата. Предоставляет дополнительные варианты сортировки и фильтрации памяти. Компонент **История сообщений** можно использовать как с языковой моделью или агентом, так и без них. Например, для извлечения данных из памяти вне контекста чата — напрямую из базы данных памяти чата без передачи в чат.
- **Компоненты сторонней памяти чата** — используйте только при необходимости хранить или извлекать память чата из отдельной внешней базы данных. Как правило, это требуется при специфических потребностях в хранении, которые не удовлетворяются хранилищем Генеративный ИИ, — например, при необходимости управлять данными памяти напрямую через базу данных или использовать другую базу данных.

Для получения дополнительной информации и примеров см. компонент **История сообщений** и Память агента.

17.3 Управление коммуникацией между компонентами через идентификатор сеанса

Идентификатор сеанса — это уникальный идентификатор соединения между клиентом и сервером. Один сеанс соответствует продолжительности соединения клиента с сервером.

В **Playground** Генеративный ИИ активные сеансы отображаются в левой части панели.

Генеративный ИИ использует идентификаторы сеансов для отслеживания различных чат-взаимодействий внутри потоков, что позволяет поддерживать несколько чат-сеансов в одном потоке. Сообщения сохраняются в базе данных вместе с соответствующими значениями `session_id`.

Разделение по сеансам полезно для управления клиент-серверными соединениями и важно для поддержания независимых контекстов диалога в рамках одного потока. LLM опираются на историю взаимодействий при генерации ответов, и если диалоги не разделены, ответы могут стать менее точными или запутанными.

Настройка идентификатора сеанса

Пользовательские идентификаторы сеансов можно задать в теле API-запроса или в расширенных настройках отдельных компонентов. Значение идентификатора сеанса из API имеет приоритет. Если идентификатор сеанса не указан, используется идентификатор потока.

Если вы задаёте пользовательский идентификатор сеанса в теле запроса, все последующие компоненты используют значение идентификатора сеанса из предыдущего компонента.

Замените `SERVER_ADDRESS`, `FLOW_ID` и `ASTRA_MULTIAGENT_AI_API_KEY` значениями из вашего развёртывания Генеративный ИИ:

```
curl --request POST \
  --url "http://SERVER_ADDRESS/api/v1/run/FLOW_ID" \
  --header "Content-Type: application/json" \
  --header "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \
  --data '{
    "input_value": "Hello",
    "output_type": "chat",
    "input_type": "chat",
```

```
"session_id": "my_custom_session_value"
}'
```

Значение `my_custom_session_value` используется во всех компонентах, которые его принимают, а сообщения, сохранённые в ходе этого потока, записываются в `astra_multiagent_ai.db` с соответствующими значениями `session_id`.

Получение сообщений из памяти по идентификатору сеанса

Чтобы получить сообщения из локальной памяти Генеративный ИИ, добавьте компонент **История сообщений** в ваш поток. Компонент принимает `session_id` в качестве параметра фильтра и автоматически использует значение идентификатора сеанса из предыдущего компонента для получения истории сообщений из хранилища.

Сообщения также можно получить через API Генеративный ИИ с помощью запроса `GET /v1/monitor/messages`, передав `session_id`. Для получения дополнительной информации см. Конечные точки мониторинга.

Для примера использования идентификаторов сеансов см. видео [Использование идентификатора сеанса в модуле «Генеративный ИИ»](#).

17.4 Управление векторными данными

Векторные данные критически важны для AI-приложений. Генеративный ИИ предоставляет несколько компонентов для хранения и извлечения векторных данных в ваших потоках: модели эмбедингов, векторные хранилища и базы знаний.

Модели эмбедингов

Компоненты модели эмбедингов генерируют векторные представления текста с помощью указанной большой языковой модели (LLM).

Существует два распространённых сценария использования этих компонентов:

- **Сохранение векторов** — генерация эмбедингов для контента, записываемого в векторную базу данных.
- **Поиск по векторам** — генерация эмбединга из запроса для выполнения поиска по сходству.

В обоих случаях компонент модели эмбедингов подключается к компоненту векторного хранилища. Для получения дополнительной информации, примеров и описания доступных параметров см. компоненты модели эмбедингов.

В качестве альтернативы вы можете использовать базы знаний, которые включают встроенную поддержку нескольких моделей эмбедингов.

Векторные хранилища

Компоненты векторного хранилища читают и записывают данные в векторные базы данных. Обычно такие компоненты подключаются к удалённым базам данных, однако некоторые поддерживают и локальные.

Векторные данные необходимы для LLM-приложений — чат-ботов и агентов.

Хотя LLM можно использовать самостоятельно для общих чат-взаимодействий и типовых задач, интеграция векторных баз данных позволяет реализовать контекстно-зависимые сценарии (например, RAG) и работу с пользовательскими наборами данных (например, внутренними бизнес-данными). Для этого часто требуется интеграция векторных баз данных и векторного поиска, которые предоставляют дополнительный контекст и формируют осмысленные запросы.

Генеративный ИИ включает компоненты векторного хранилища для чтения и записи векторных данных: хранение эмбеддингов, поиск по сходству, обходы Graph RAG и специализированные поисковые реализации, такие как OpenSearch. Из-за взаимозависимой функциональности компоненты векторного хранилища, языковой модели и модели эмбеддингов обычно используются совместно в одном потоке или в серии взаимосвязанных потоков.

Чтобы найти доступные компоненты векторного хранилища, просмотрите раздел **Пакеты** или воспользуйтесь **поиском** по предпочтительному провайдеру векторной базы данных.

Пример: поток векторного поиска

Руководство по использованию векторных данных в потоке см. в разделе Создать чат-бот с векторным RAG. Следующий пример демонстрирует использование компонентов векторного хранилища в потоках совместно с сопутствующими компонентами — моделью эмбеддингов и языковой моделью. Описанные шаги охватывают важные детали конфигурации, функциональность и рекомендации по эффективному использованию этих компонентов. Это лишь один из возможных сценариев; пример не является исчерпывающим руководством.

1. Создайте поток на основе шаблона **Векторное хранилище RAG**.

Шаблон содержит два подпотока. Подпоток **Загрузка данных** загружает эмбеддинги и контент в векторную базу данных, а подпоток **Поисковый модуль** выполняет векторный поиск для получения релевантного контекста на основе запроса пользователя.

2. Настройте подключение к базе данных для обоих компонентов **Astra DB** или замените их другой парой компонентов векторного хранилища. Убедитесь, что компоненты подключаются к одному и тому же векторному хранилищу и что компонент в подпотоке **Поисковый модуль** поддерживает поиск по сходству.

Параметры каждого компонента векторного хранилища зависят от его роли в потоке. В этом примере подпоток **Загрузка данных** *записывает* в векторное хранилище, а подпоток **Поисковый модуль** *читает* из него. Следовательно, параметры поиска актуальны только для компонента **Векторный поиск** в подпотоке **Поисковый модуль**. Описание конкретных параметров см. в документации выбранного компонента векторного хранилища.

3. Чтобы настроить модель эмбеддингов, выполните одно из следующих действий:

- **Использовать модель OpenAI** — в обоих компонентах **OpenAI Эмбеддинги** введите ваш ключ API OpenAI. Вы можете использовать модель по умолчанию или выбрать другую модель эмбеддингов OpenAI.
- **Использовать другого провайдера** — замените компоненты **OpenAI Эмбеддинги** другой парой компонентов модели эмбеддингов и настройте параметры и учётные данные соответственно.
- **Использовать Astra DB vectorize** — если вы используете векторное хранилище Astra DB с интеграцией vectorize, оба компонента **OpenAI Эмбеддинги** можно удалить. В этом случае интеграция vectorize автоматически генерирует эмбеддинги из **Загружаемых данных** (подпоток **Загрузка данных**) и **Поискового запроса** (подпоток **Поисковый модуль**).

Если ваше векторное хранилище уже содержит эмбеддинги, убедитесь, что компоненты модели эмбеддингов используют ту же модель, что применялась

ранее. Смешивание моделей эмбеддингов в одном хранилище может привести к неточным результатам поиска.

4. В компоненте **Разделить текст** оптимизируйте настройки разбиения для вашей модели эмбеддингов. Например, если у модели эмбеддингов ограничение в 512 токенов, параметр **Размер фрагмента** не должен превышать это значение.

Поскольку подпоток **Поисковый модуль** передаёт ввод чата напрямую в компонент векторного хранилища для поиска, убедитесь, что строка ввода чата не превышает ограничений вашей модели эмбеддингов. В тестовых целях можно вводить запросы, укладывающиеся в ограничения; однако в рабочей среде может потребоваться реализовать дополнительные проверки или этапы предобработки. Например, используйте дополнительные компоненты для подготовки ввода чата перед выполнением векторного поиска или ограничьте ввод в коде приложения.

4. В компоненте **Языковая модель** введите ваш ключ API OpenAI или выберите другого провайдера и модель для части потока, отвечающей за чат.
5. Запустите подпоток **Загрузка данных** для заполнения векторного хранилища. В компоненте **Чтение файла** выберите один или несколько файлов, а затем нажмите **Run component** на компоненте векторного хранилища в подпотоке **Загрузка данных**.

Подпоток **Загрузка данных** загружает файлы с локального компьютера, разбивает их на фрагменты, генерирует эмбеддинги для этих фрагментов и сохраняет фрагменты вместе с их эмбеддингами в векторной базе данных.

Подпоток **Загрузка данных** отделён от подпотока **Поисковый модуль**, поскольку загружать данные при каждом чат-взаимодействии не требуется. Запускайте подпоток **Загрузка данных** по мере необходимости — для первичной загрузки или обновления данных в хранилище. Взаимодействие в чате использует только компоненты, необходимые для чата.

Если ваше векторное хранилище уже содержит нужные данные, запускать подпоток **Загрузка данных** не требуется.

6. Откройте **Playground** и начните чат, чтобы запустить подпоток **Поисковый модуль**. Подпоток **Поисковый модуль** генерирует эмбеддинг из ввода чата, выполняет векторный поиск для получения похожего контента из хранилища, разбирает результаты поиска в дополнительный контекст для LLM, а затем использует LLM для генерации ответа на естественном языке. LLM использует результаты векторного поиска совместно со своими обучающими данными и инструментами — например, базовым веб-поиском и сведениями о дате и времени — для формирования ответа. Чтобы избежать передачи полного блока необработанных результатов поиска в LLM, компонент **Парсер** извлекает строки text из объекта результатов поиска Data и передаёт их в компонент **Шаблон промпта** в формате Message. Затем строки и остальное содержимое шаблона компилируются в инструкции на естественном языке для LLM. Для этого преобразования вы можете использовать и другие компоненты — например, **Операции с данными** — в зависимости от того, как вы хотите использовать результаты поиска.

Чтобы просмотреть необработанные результаты поиска, нажмите **Выход** на компоненте векторного хранилища после запуска подпотока **Поисковый модуль**.

Базы знаний

База знаний Генеративный ИИ — это локальная векторная база данных, хранящаяся в хранилище Генеративный ИИ.

Поскольку базы знаний являются локальными, данные не запрашиваются удалённо и не переобрабатываются при каждом запуске потока. Это может быть эффективнее, чем использование удалённой векторной базы данных, и хорошо подходит для потоков, работающих с пользовательскими предметно-ориентированными наборами данных — например, фрагментами данных о клиентах и продуктах.

Компоненты баз знаний можно использовать практически так же, как компоненты векторных хранилищ. Однако между ними есть ряд ключевых отличий:

- **Локальное хранение** — базы знаний Генеративный ИИ хранятся исключительно локально. Только некоторые компоненты векторных хранилищ поддерживают локальные базы данных.
- **Встроенные модели эмбедингов** — базы знаний Генеративный ИИ включают встроенную поддержку нескольких моделей эмбедингов. Другие модели с базами знаний не поддерживаются. Чтобы использовать другого провайдера или модель, применяйте компонент векторного хранилища совместно с нужным компонентом модели эмбедингов.
- **Базовый поиск по сходству** — базы знаний Генеративный ИИ поддерживают только стандартный поиск по сходству. Для более сложных запросов используйте компонент векторного хранилища, подключённого к провайдеру с необходимой функциональностью.
- **Структурированные данные** — базы знаний Генеративный ИИ поддерживают только структурированные данные. Для неструктурированных данных используйте совместимый компонент векторного хранилища.

Расположение хранилища баз знаний

Каждая база знаний представляет собой векторную базу данных ChromaDB. Каждая база данных хранится в отдельном каталоге, который содержит:

- **Векторные эмбединги** — эмбединги, хранящиеся в векторной базе данных Chroma.
- **Файлы метаданных** — конфигурация и информация о модели эмбедингов.
- **Исходные данные** — данные, использованные для создания базы знаний.

Базы знаний хранятся локально в вашем экземпляре Генеративный ИИ. Расположение по умолчанию зависит от операционной системы и способа установки:

- **Генеративный ИИ Desktop:**
 - **macOS:** /Users/<username>/AstraMultiagentAI/knowledge_bases
 - **Windows:**
C:\Users\<username>\AppData\Roaming\com.AstraMultiagentAIDesktop\knowledge_bases
- **Генеративный ИИ OSS:**
 - **macOS/Windows/Linux/WSL** (uv pip install): <path_to_venv>/lib/python3.12/site-packages/AstraMultiagentAI/knowledge_bases (версия Python может различаться; базы знаний не являются общими между виртуальными окружениями)
 - **macOS/Windows/Linux/WSL** (git clone):
<path_to_clone>/src/backend/base/AstraMultiagentAI/knowledge_bases

Если задана переменная окружения ASTRA_CONFIG_DIR, подкаталог knowledge_bases будет создан относительно указанного пути.

Чтобы изменить путь к каталогу knowledge_bases по умолчанию, задайте переменную окружения ASTRA_KNOWLEDGE_BASES_DIR:

Создание базы знаний

В этом примере вы создадите базу знаний из фрагментированных заказов клиентов. Чтобы следовать примеру, скачайте customer-orders.csv.

1. На странице **Проекты** нажмите **Знания** под списком проектов для просмотра и управления базами знаний.
2. Чтобы создать новую базу знаний, нажмите **Добавить знания**.
3. В панели **Создание базы знаний** введите название базы знаний и выберите модель эмбедингов.

Чтобы изменить глобальную конфигурацию провайдеров моделей Генеративный ИИ:

1. Нажмите на значок профиля, выберите **Настройки**, а затем **Провайдеры моделей**.
2. На панели **Провайдеры моделей** выберите нужного провайдера.
3. В поле **Ключ API** введите ключ API вашего провайдера.

Ключ должен иметь разрешение на вызов моделей, которые вы планируете использовать, а ваш аккаунт должен располагать достаточным количеством кредитов. Для каждого провайдера можно добавить только один ключ. Убедитесь, что ключ предоставляет доступ ко *всем* моделям, которые вы хотите использовать в модуле «Генеративный ИИ».

4. Включите конкретные модели, которые вы хотите использовать в модуле «Генеративный ИИ». Доступные модели зависят от провайдера и прав вашего ключа API. Текстовые модели перечислены в разделе **Языковые модели**. Модели, генерирующие эмбединги, перечислены в разделе **Модели эмбедингов**.

После включения модели в глобальной конфигурации вы сможете использовать её в любом модельном компоненте ваших потоков.

5. Чтобы настроить источники для базы знаний, нажмите **Настройка источников**. Если вы хотите создать пустую базу знаний, нажмите **Создать**.
6. На панели **Настройка источников** задайте источники данных и параметры разбиения для векторного поиска. Для этого примера нажмите **Добавить источник** и выберите загруженный файл customer-orders.csv. Настройки по умолчанию для параметров **Размер фрагмента**, **Перекрытие фрагментов** и **Разделитель** подходят. Чтобы продолжить, нажмите **Следующий шаг**.
7. Панель **Обзор и сборка** позволяет просмотреть первый фрагмент перед тем, как тратить токены на встраивание всех данных. Если фрагмент не соответствует требованиям, нажмите **Назад** для корректировки стратегии разбиения. Чтобы встроить данные, нажмите **Создать**.
8. Ваши данные встроены как **Знания**. Когда они будут готовы к использованию, статус в столбце **Статус** изменится на **Готов**.

Чтобы использовать новую базу знаний в потоке, см. Использование компонента «База знаний» в потоке.

Управление базами знаний

На странице **Проекты** нажмите **Знания** под списком проектов для просмотра и управления базами знаний.

Для каждой базы знаний отображается следующая информация:

- Название
- Модели эмбедингов

- Размер на диске
- Количество слов, символов и фрагментов
- Средняя длина и размер фрагментов
- Статус базы знаний

Поведение разбиения определяется моделью эмбедингов, которая задаётся при создании базы знаний. Если необходимо изменить модель эмбедингов, базу знаний потребуется удалить и создать заново.

Чтобы обновить базу знаний, нажмите **Ещё**, а затем выберите **Обновить базу знаний**.

Чтобы просмотреть фрагменты базы знаний, нажмите **Ещё**, а затем выберите **Просмотреть фрагменты**.

Чтобы удалить базу знаний, нажмите **Ещё**, а затем **Удалить**. Если какие-либо потоки используют удалённую базу знаний, их необходимо обновить для использования другой базы.

Для получения дополнительной информации об использовании баз знаний в потоке см. документацию компонента **База знаний**.

18. Наблюдаемость

18.1 Журналы

Генеративный ИИ ведёт журналы для отдельных потоков и самого приложения, используя библиотеку `structlog`.

Основной файл журнала по умолчанию называется Генеративный ИИ.log.

Файлы журналов хранятся в формате JSON со структурированными метаданными.

Просмотр журналов в реальном времени

Чтобы отслеживать журналы Генеративный ИИ по мере их создания, следите за файлом журнала:

1. Перейдите в каталог конфигурации Генеративный ИИ:

```
cd /Users/USERNAME/Library/Caches/AstraMultiagentAI
cd C:\Users\USERNAME\AppData\Local\com.AstraMultiagentAIDesktop\logs
```

2. Отслеживайте основной файл журнала:

```
tail -f Генеративный ИИ.log
Get-Content -Wait -Path "Генеративный ИИ.log"
```

Если новые записи не появляются, убедитесь, что Генеративный ИИ запущен, и выполните какие-либо действия для генерации событий журнала. Также проверьте терминал, в котором был запущен Генеративный ИИ, — журналы могут выводиться там.

Журналы потоков и компонентов

После запуска потока вы можете просмотреть журналы для каждого компонента и каждого выполнения потока. Например, можно просмотреть объекты `Message`, полученные и сгенерированные компонентами ввода и вывода.

Просмотр журналов потока

В визуальном редакторе нажмите **Logs**, чтобы просмотреть журналы всего потока.

Затем нажмите на ячейки в столбцах **входы** и **выходы**, чтобы просмотреть объекты `Message`. Например, следующие данные `Message` могут быть выводом компонента **Ввод в чат**:

```
"messages": [
  {
    "message": "What's the recommended way to install Docker on Mac M1?",
    "sender": "User",
    "sender_name": "User",
    "session_id": "Session Apr 21, 17:37:04",
    "stream_url": null,
    "component_id": "ChatInput-4WKag",
    "files": [],
    "type": "text"
  }
]
```

В случае компонентов ввода/вывода исходные данные могут не иметь структуру объекта `Message`. Например, языковая модель может передать необработанный текстовый ответ компоненту **Вывод чата**, который затем преобразует его в объект `Message`.

Файлы `.log` для потоков находятся в каталоге хранения журналов вашей установки Генеративный ИИ. Пути к файлам см. в разделе [Хранение журналов](#).

Просмотр журналов чата

В **Playground** можно просмотреть историю чата для каждой сессии. Для получения дополнительной информации см. [Просмотр истории чата](#).

Просмотр вывода отдельного компонента

При отладке проблем с форматом или содержимым вывода потока полезно просматривать вывод каждого компонента, чтобы определить, где данные теряются или искажаются.

Чтобы просмотреть вывод, созданный отдельным компонентом в последнем запуске, нажмите **Inspect output** на компоненте в визуальном редакторе.

18.2 Трассировки

Функция **Трассировки** в модуле «Генеративный ИИ» записывает детальные трассировки выполнения ваших потоков и компонентов, позволяя отлаживать проблемы, измерять задержку и отслеживать использование токенов без подключения внешних систем наблюдения.

Данные трассировок хранятся в базе данных Генеративный ИИ в таблицах `traces` и `span`. Они отображаются на страницах **Активность потока** и **Детали трассировки** в пользовательском интерфейсе, а также доступны через API-конечную точку `/monitor/traces`.

Трассировки включены по умолчанию. Чтобы отключить встроенную трассировку и использовать стороннего поставщика, установите переменную `ASTRA_MULTIAGENT_AI_NATIVE_TRACING` в значение `false`.

Что захватывают трассировки

Трассировщик записывает:

- **Трассировки уровня потока:** трассировка для каждого запуска потока, включающая общее время выполнения и статус.
- **Спаны компонентов:** спан для каждого компонента в потоке, включающий входные данные, выходные данные, задержку и ошибки.
- **Спаны LangChain:** более детальные спаны для цепочек, инструментов, ретриверов и вызовов LLM, в том числе название модели и использование токенов, если доступно.

Каждый спан включает:

- **Название и Тип** (например, `chain`, `llm`, `tool`, `retriever`)
- **Время начала и окончания** и задержку (мс)
- **Входные и выходные данные** (сериализованные)
- **Сведения об ошибке**, если спан завершился с ошибкой
- **Атрибуты**, такие как количество токенов и метаданные модели

Просмотр трассировок в UI

Чтобы просмотреть трассировки в пользовательском интерфейсе Генеративный ИИ:

1. Запустите поток, например стартовый поток **Simple Agent** из руководства Быстрый старт.
2. Нажмите **Трассировки**. Откроется страница **Активность потока**. Каждый запуск потока отображается как единая трассировка всех его спанов. Запуски можно дополнительно фильтровать по идентификатору сеанса, статусу или временному диапазону. При желании нажмите **Скачать**, чтобы сохранить JSON-файл трассировки потока на локальный компьютер.
3. Нажмите на запуск потока, чтобы открыть панель **Детали трассировки**. Панель отображает спаны запуска потока, включая общий спан всего запуска и отдельные спаны для каждого компонента. Спаны компонентов включают входные и выходные данные, время выполнения и использование токенов.

Получение трассировок через API

Чтобы запросить трассировки программно, используйте конечные точки `/monitor/traces`. Полное описание параметров и примеры кода на Python, TypeScript и curl см. в разделе Конечные точки мониторинга: `Get traces`.

19. Типы данных

Компоненты Генеративный ИИ разработаны для приема и выдачи определённых типов входных и выходных данных. Типы входных и выходных данных определяют структуру и поток информации между компонентами. Понимание этих структур помогает создавать приложения, которые предоставляют корректный ввод и правильно предсказывают формат вывода.

Порты компонентов представляют типы данных, которые каждый компонент может отправлять и получать. Некоторые типы данных очевидны из полей, к которым они привязаны; например, поле **Системное сообщение** принимает данные сообщения. Цвета портов также указывают тип данных порта. Например, порты **Данные**, представленные , могут принимать или выдавать структурированные объекты данных.

При построении потоков соединяйте выходные порты с входными портами одного и того же типа (цвета), чтобы передавать этот тип данных между двумя компонентами.

- В рабочей области наведите курсор на порт, чтобы увидеть детали соединения для этого порта. Нажмите на порт, чтобы **выполнить поиск** совместимых компонентов.
- Если два компонента имеют несовместимые типы данных, вы можете использовать компонент обработки, такой как компонент **Конвертировать Тип**, чтобы преобразовать данные между компонентами.

19.1 Данные

Порты **данных** принимают или создают тип **Данные**, который представляет собой структурированный объект данных, например JSON-полезную нагрузку, которую вы можете отправить по API. Этот тип данных используется для передачи пар «ключ-значение» между компонентами, например профилей пользователей, настроек или другой структурированной информации.

Объекты типа **Данные** включают основное текстовое, обозначенное `text_key`, и дополнительные метаданные.

Схема и атрибуты

Схема определена в `data.py`.

Доступны следующие атрибуты:

- **Данные:** Объект **Данные** хранит пары «ключ-значение» в атрибуте `.data`. Это основной словарь объекта **Данные**. Каждый ключ — это имя поля, а значения могут быть любого поддерживаемого типа данных.
- `text_key`: Ключ в объекте **Данные**, который считается основным текстовым значением.
- `default_value`: Запасной вариант, если `text_key` отсутствует. Значение по умолчанию для `text_key` — `"text"`.

```
data_obj = Data(
    text_key="text",
    data={
        "text": "Hello world",
        "name": "Charlie",
        "age": 28
    },
    default_value=""
)
```

Объекты Данные могут быть сериализованы в JSON, созданы из JSON или созданы из других словарных данных. Однако полученный объект Данные является структурированным объектом с валидацией и методами, а не простым словарём. Например, при сериализации в JSON предыдущий пример на Python превращается в следующий JSON-объект:

```
{
  "text_key": "text",
  "data": {
    "text": "Hello world",
    "name": "Charlie",
    "age": 28
  },
  "default_value": ""
}
```

19.2 DataFrame

DataFrame порты принимают или создают pandas DataFrames, которые похожи на табличные CSV данные.

Используйте тип DataFrame, чтобы работать с данными, содержащими несколько строк или записей.

Схема и атрибуты

Схема определена в dataframe.py.

Доступны следующие атрибуты:

- **Полная совместимость с pandas:** Поддерживаются все методы и функции pandas DataFrame
- **Интеграция с модулем «Генеративный ИИ»:** Принимает списки объектов Данные, словари или существующие DataFrame.
- **Удобные методы:**
 - to_data_list()
 - add_row()
 - add_rows()
 - to_lc_documents()
 - to_data()
 - to_message()
- **Поддержка ключа Текст:** Сохраняет атрибуты text_key и default_value для совместимости с объектом Данные.

Структура DataFrame

DataFrame имеет табличную структуру данных с строками и столбцами. Ключи — это столбцы, а каждый объект в массиве представляет собой строку.

```
[
  {
    "name": "Charlie Lastname",
    "age": 28,
    "email": "charlie.lastname@example.com"
  },
  {
    "name": "Alexandra Example",
    "age": 34,
    "email": "alexandra@example.com"
  }
]
```

При представлении в виде табличных данных объект DataFrame выглядит следующим образом:

```
| name | age | email |
|-----|-----|
| Charlie Lastname | 28 | charlie.lastname@example.com |
| Alexandra Example | 34 | alexandra@example.com |
```

19.3 Эмбединги

Эмбединг порты генерируют или принимают векторные эмбединги для поддержки функций, таких как поиск по сходству.

Тип данных Embeddings используется в основном компонентами, которые либо создают, либо потребляют векторные эмбединги, такими как компоненты моделей эмбедингов и компоненты векторных хранилищ.

Например, компоненты моделей эмбедингов выводят данные типа Embeddings, которые можно подключить к входному порту компонента **Эмбединги** векторного хранилища

Для получения информации о базовых классах Python, которые создают Embeddings, см. документацию моделей Эмбединги LangChain.

19.4 LanguageModel

Тип Языковая модель — это особый тип данных, который может быть произведён компонентами языковой модели и принят компонентами, использующими LLM.

При изменении типа вывода компонента языковой модели с **Ответ модели** на **Языковая модель**, выходной порт компонента меняется с порта **Сообщения** на порт **Языковая модель**. Затем вы подключаете исходящий порт **Языковая модель** к входному порту совместимого с **Языковая модель** компонента, например, к компоненту **Умное преобразование**.

Для получения дополнительной информации об использовании этих компонентов в потоках и переключении вывода Языковая модель, см. компоненты моделей Язык.

LanguageModel является экземпляром LangChain ChatModel.

Поскольку Генеративный ИИ построен на основе LangChain, Языковая модель на самом деле является экземпляром модели чата LangChain, использующей параметры конфигурации, заданные в исходном компоненте.

Часто компоненты создают экземпляр интегрированной модели чата, предназначенной для поддержки конкретного поставщика модели, например, ChatOpenAI или ChatAnthropic.

Вы можете просмотреть код компонента, чтобы узнать какой именно экземпляр Chat он создаёт.

19.5 Память

Порты **Память** используются для интеграции компонента **История сообщений** с внешним хранилищем памяти чата.

Для получения дополнительной информации см. компонент **История сообщений**.

19.6 Сообщения

Порты **Сообщения** принимают или генерируют Сообщение данные, которые расширяют тип Данные дополнительными полями и методами для ввода Текст, обычно используемого в чат-потоках.

Этот тип данных используется многими компонентами.

Компоненты, которые принимают или генерируют данные типа Сообщение, могут не включать все атрибуты во входящих или исходящих данных Сообщение. Данные могут считаться корректными, если они соответствуют схеме Сообщение.

При построении потоков обращайте внимание на поля, отображаемые в каждом компоненте в рабочей области, а не на типы данных, передаваемых между компонентами. Сведения о конкретном типе данных часто имеют значение только при отладке потока или компонента, который не выдаёт ожидаемый результат.

Например, компонент **Ввод чата** требует только содержимое поля **Ввод текста** (input_value). Затем компонент создает полный объект Сообщение перед передачей данных другим компонентам в потоке.

Схема, структура и атрибуты

Схема Сообщение определена в message.py. Некоторые атрибуты Сообщение имеют собственные определения схем, такие как content_block.py.

Данные Сообщение структурированы как объект JSON. Например:

```
json showLineNumbers {      "text": "Name: Charlie Lastname, Age: 28, Email:
charlie.lastname@example.com",  "sender": "User",    "sender_name": "Charlie Lastname",
"session_id": "some-session-id",  "timestamp": "2024-06-01T12:00:00Z",    "files": [],
"content_blocks": [], "category": "message" }
```

Включаемые атрибуты в конкретном объекте Сообщение зависят от контекста, включая тип компонента, активность потока и то, является ли сообщение запросом или ответом. Некоторые общие атрибуты включают следующее:

- text: Основное содержание сообщения.
- sender: Определяет отправителя чат-сообщения как Пользователь или Языковая модель.
- sender_name: Отображаемое имя отправителя. По умолчанию Пользователь или Языковая модель.
- session_id: Идентификатор сессии чата.
- flow_id: Идентификатор потока, с которым связано сообщение. flow_id и session_id одинаковы, если поток не использует пользовательские идентификаторы сессий.
- timestamp: Временная метка UTC, когда сообщение было отправлено.
- files: Список путей к файлам или изображений, включенных в сообщение.
- content_blocks: Контейнер для ввода богатого контента, такого как текст, медиафайлы или код. Также содержит информацию об ошибке, если LLM не может обработать ввод.
- category: "message", "error", "warning" или "info".

Не все атрибуты обязательны, и некоторые компоненты принимают совместимый с сообщением ввод, например необработанный текстовый ввод. Строгость зависит от компонента.

Сообщения данные в компонентах Ввод и Вывод

В потоках с компонентами **Ввод и вывод чата**, данные типа Сообщение обеспечивают согласованную структуру для взаимодействий в чате и идеально подходят для чат-ботов, анализа диалогов и других сценариев, основанных на диалоге с LLM или агентом. В этих потоках чат-интерфейс **Песочница** выводит только атрибуты Сообщение, релевантные разговору, такие как text, files и сообщения об ошибках от content_blocks. Чтобы увидеть все атрибуты Сообщение, просмотрите журналы сообщений в **Песочница**.

В потоках с компонентами **Текстовый ввод и вывод** данные Сообщение используются для передачи простых текстовых строк без метаданных, связанных с чатом. Эти компоненты обрабатывают данные Сообщение как независимые текстовые строки, а не как часть текущего диалога. По этой причине поток, содержащий только **Текстовый ввод и вывод** компоненты, не совместим с **Песочница**. Для получения дополнительной информации см. **Текстовый ввод и вывод**.

При использовании API Генеративный ИИ ответ включает объект Сообщение вместе с другими данными ответа от выполнения потока. Ответы API Генеративный ИИ могут быть чрезвычайно подробными, поэтому ваши приложения должны включать код для извлечения релевантных данных из ответа, чтобы вернуть их пользователю. Для примера см. Генеративный ИИ quickstart.

Кроме того, ввод, отправляемый в порт входа компонентов ввода/вывода, *не* обязательно должны быть полным объектом Сообщение, поскольку компонент сам создает объект Сообщение, который затем передается другим компонентам в потоке или возвращается как вывод потока. Фактически, некоторые компоненты не должны получать полный объект Сообщение, поскольку такие атрибуты, как timestamp, должны быть добавлены компонентом для точности.

19.7 Инструмент

Порты **Инструмент** соединяют инструменты с компонентом **Агент**.

Инструменты могут быть другими компонентами, где вы включили **Инструмент режима**, они могут быть выделенным компонентом **Инструменты MCP**, или они могут быть другими компонентами, которые поддерживают только **Инструмент режима**. Несколько инструментов могут быть подключены к одному и тому же компоненту **Агент** через один и тот же порт.

Функционально данные Инструмент представляют собой объект LangChain StructuredTool, который может использоваться в потоках агентов.

Для получения дополнительной информации см. Configure tools for agents и Use Генеративный ИИ as an MCP client.

19.8 Неизвестные или множественные типы

Если Порт может принимать или генерировать несколько типов данных, он отображается серой иконкой порта .

Наведите курсор на Порт, чтобы увидеть принимаемые или генерируемые типы данных.

19.9 Просмотр типов данных в потоках

В Генеративный ИИ вы можете использовать **Исследовать вывод** для просмотра вывода отдельных компонентов. Это поможет вам узнать о различных типах данных и отладить проблемы с недействительными или некорректными вводами и выводом.

Следующий пример показывает, как просмотреть вывод компонента **Конвертировать Тип**, который может преобразовывать данные из одного типа в другой:

1. Создайте поток, а затем подключите компонент **Ввод чата** к компоненту **Конвертировать Тип**.
2. В компоненте **Ввод чата** введите текст для обработки конвертером типов.
3. В компоненте **Конвертировать Тип** нажмите **Запустить компонент**, а затем нажмите **Исследовать вывод**.

Вывод по умолчанию — данные Сообщение, которые совпадают с вводом, поступающим от компонента **Ввод чата**. Чтобы увидеть данные Сообщение, преобразованные в Данные или DataFrame, измените **Тип выхода** в компоненте **Конвертировать Тип**, а затем повторно запустите компонент.

- Сообщение
- Данные
- DataFrame

Input text

```
{
  "timestamp": "2025-07-15 20:56:20 UTC",
  "sender": "User",
  "sender_name": "User",
  "session_id": "a0c7e888-4fd6-4242-b8c8-e761ad690aeb",
  "text": "",
  "files": [],
  "error": false,
  "edit": false,
  "properties": {
    "text_color": "",
    "background_color": "",
    "edited": false,
    "source": {
      "id": null,
      "display_name": null,
      "source": null
    },
  },
  "icon": "",
  "allow_markdown": false,
  "positive_feedback": null,
  "state": "complete",
  "targets": []
},
"category": "message",
"content_blocks": [],
"id": "9da72da2-efbb-4ccd-90ad-b32429b0418e",
"flow_id": "a0c7e888-4fd6-4242-b8c8-e761ad690aeb",
"duration": null
}
```

Field	Value
timestamp	2025-07-15 20:56:11 UTC
sender	User
sender_name	User
session_id	a0c7e888-4fd6-4242-b8c8-e761ad690aeb

```
| text | (empty) |
| files | [] |
| error | False |
| edit | False |
| properties | text_color: ", background_color: ", edited: False, source: {id: None, display_name: None, source: None},
icon: ", allow_markdown: False, positive_feedback: None, state: 'complete', targets: [] |
| category | message |
| content_blocks | [] |
| id | 341686eb-7a39-4b80-a90a-d8bf267815ef |
| flow_id | a0c7e888-4fd6-4242-b8c8-e761ad690aeb |
| duration | (empty) |
```

20. Голосовой режим

Голосовой режим недоступен в модуле «Генеративный ИИ» Desktop. Чтобы использовать голосовой режим, установите пакет Генеративный ИИ OSS Python. Голосовой режим Генеративный ИИ позволяет взаимодействовать с потоками через микрофон и динамики.

20.1 Тестирование голосового режима в Playground

В **Playground** нажмите кнопку **Микрофон**, чтобы включить голосовой режим и взаимодействовать с потоками голосом.

Следующие шаги демонстрируют включение голосового режима на основе шаблона **Простой агент**:

1. Создайте поток на основе шаблона **Простой агент**.
2. Добавьте **API-ключ OpenAI** в компонент **Агент**.
3. Нажмите **Playground**.
4. Нажмите значок **Микрофон**, чтобы открыть диалог **Голосовой режим**.
5. Введите API-ключ OpenAI и нажмите **Сохранить**. Генеративный ИИ сохраняет ключ как глобальную переменную.
6. Если появится запрос на доступ к микрофону, разрешите его — без этого голосовой ввод недоступен.
7. В поле **Аудиовход** выберите устройство ввода для голосового режима.
8. Опционально: добавьте API-ключ ElevenLabs для использования дополнительных голосов в ответах LLM. Генеративный ИИ сохраняет этот ключ как глобальную переменную.
9. В поле **Предпочтительный язык** выберите язык для диалогов с LLM. Настройка влияет как на язык ввода, так и на язык ответа.
10. Говорите в микрофон, чтобы начать чат.

Если всё настроено правильно, волновая форма фиксирует ваш ввод, логика агента выполняется, а ответ воспроизводится голосом и отображается в **Playground**.

23. Безопасность

Интерфейс Генеративный ИИ представляет собой IDE и платформу выполнения кода, что означает, что Генеративный ИИ по своей природе способен исполнять произвольный код, предоставленный разработчиком. UI Генеративный ИИ включает редактор кода, позволяющий разработчикам писать и выполнять произвольный Python с полным доступом к процессу бэкенда Генеративный ИИ, файловой системе и сети. Кроме того, некоторые компоненты используют выполнение кода, в том числе генерацию кода моделями ИИ на основе пользовательского ввода с последующим его исполнением.

Генеративный ИИ не обеспечивает изоляцию между пользователями в рамках одного процесса и не ограничивает доступ к локальному диску или сетевым ресурсам. Видимость потоков и контроль доступа пользователей разработаны для удобства использования, а не для обеспечения безопасности. Пользователи могут напрямую обращаться к базовым соединениям с базой данных и системным ресурсам. Для многопользовательских развёртываний Генеративный ИИ полагается на безопасность на уровне инфраструктуры, а не на изоляцию на уровне приложения — ответственность за обеспечение изоляции арендаторов лежит на вас.

Вы несёте ответственность за следующее:

- Обеспечение комплексной изоляции инфраструктуры.
- Безопасный запуск потоков, особенно тех, которые используют код, сгенерированный LLM, или пользовательский ввод, который может содержать произвольный код.
- Обеспечение сквозной безопасности взаимодействий конечных пользователей с API на базе Генеративный ИИ, включая защиту данных в состоянии покоя и при передаче.

23.1 Безопасность собственных развёртываний

Когда вы предоставляете API на основе потоков, созданных вами или вашей организацией, вы несёте ответственность за защиту конечных пользователей этих API.

Следуйте отраслевым лучшим практикам для API:

- Используйте безопасный API-шлюз для обеспечения аутентификации и авторизации.
- Обеспечьте надлежащую изоляцию пользовательских данных.
- Выполняйте санитизацию входных и выходных данных для защиты от XSS и атак внедрения, включая регулярные выражения для предотвращения уязвимостей ReDoS.

Для получения дополнительной информации о настройке обратного прокси см. Развёртывание Генеративный ИИ с Nginx и SSL. Для получения дополнительной информации о конфигурации аутентификации см. API-ключи и аутентификация.

23.2 Безопасность сторонних развёртываний

Если вы предоставляете Генеративный ИИ как сервис третьим сторонам, следует исходить из того, что любой код, исполняемый Генеративный ИИ, потенциально может быть вредоносным.

Генеративный ИИ не обеспечивает изоляцию между арендаторами, поэтому ответственность за её реализацию на уровне инфраструктуры лежит на вас:

- Используйте изоляцию на уровне процессов, чтобы несколько арендаторов не использовали один процесс Генеративный ИИ.
- Используйте изоляцию на уровне диска, чтобы предотвратить совместный доступ к записываемому постоянному хранилищу.
- Используйте изоляцию на уровне сети, чтобы предотвратить доступ к частным сетям.
- Используйте изоляцию на уровне базы данных, чтобы предотвратить доступ к общим ресурсам базы данных или их изменение.

Аутентификация и авторизация должны обеспечиваться и применяться за пределами контейнера Генеративный ИИ. Для общих сервисов, таких как базы данных, ограничивайте доступ внешними средствами: учётными данными и политиками безопасности.

Для получения дополнительной информации см. Лучшие практики Генеративный ИИ в Kubernetes.

23.3 Бюллетень безопасности

Для получения актуальной информации об уязвимостях, исправлениях и CVE см. политику безопасности Генеративный ИИ и рекомендации по безопасности на GitHub.

Если вы обнаружили уязвимость в модуле «Генеративный ИИ», команда призывает вас ответственно сообщить о ней через вкладку Security на GitHub. Пожалуйста, не раскрывайте уязвимости публично до их оценки и устранения.

24. Обзор компонентов

Компоненты — это строительные блоки ваших потоков. Как классы в приложении, каждый компонент предназначен для конкретного сценария использования или интеграции.

Генеративный ИИ поддерживает сочетания клавиш в рабочей области.

В заголовке Генеративный ИИ нажмите значок профиля, выберите **Настройки**, затем нажмите **Сочетания клавиш**, чтобы просмотреть доступные варианты.

24.1 Добавление компонента в поток

Чтобы добавить компонент в поток, перетащите его из меню **Основные компоненты** или **Пакеты** в рабочее пространство.

Компоненты сгруппированы по типу или провайдеру, а некоторые скрыты по умолчанию:

- **Основные компоненты:** базовые компоненты Генеративный ИИ, сгруппированные по назначению, например **Входные и выходные** или **Данные**. Предоставляют общую функциональность — циклы, парсинг — а также компоненты с поддержкой нескольких сторонних интеграций.
- **Пакеты:** содержат один или несколько компонентов для конкретных сторонних интеграций, сгруппированных по провайдеру.
- **Устаревшее:** эти компоненты скрыты по умолчанию. Подробнее см. в разделе **Устаревшие компоненты**.

Настройка компонента

После добавления компонента в поток настройте его параметры и подключите к другим компонентам.

Каждый компонент имеет входы, выходы, параметры и элементы управления, соответствующие его назначению. По умолчанию отображаются только обязательные и основные параметры. Для доступа к дополнительным настройкам, включая метаметры, используйте панель инспекции компонента.

Панель инспекции компонента

При выборе компонента в рабочем пространстве справа открывается панель инспекции.

Панель инспекции отображает все параметры компонента, включая скрытые и расширенные.

Меню заголовка компонента

Чтобы открыть меню заголовка компонента, щёлкните по нему в рабочем пространстве.

В меню заголовка доступны следующие действия:

- **Код:** изменение параметров компонента путём редактирования его кода на Python.
- **Заморозить:** заморозка компонента и всех предшествующих компонентов для предотвращения повторного запуска. Подробнее см. в разделе **Заморозка компонента**.
- **Режим инструмента:** включите этот режим при использовании компонента совместно с компонентом **Агент**.

Для остальных действий, включая **Удалить** и **Дублировать**, нажмите **Показать ещё**.

Переименование компонента

Чтобы изменить имя или описание компонента, щёлкните по нему в рабочем пространстве и нажмите **Редактировать**. Описания компонентов поддерживают синтаксис Markdown.

Запуск компонента

Чтобы запустить отдельный компонент, нажмите **Запустить компонент**. Метка **Последний запуск** указывает на успешное выполнение.

Запуск отдельного компонента отличается от запуска всего потока. При запуске отдельного компонента вызывается функция `build_vertex`, которая собирает и запускает только этот компонент с входными данными, переданными через визуальный редактор (параметр `inputs_dict`). Данные `VertexBuildResult` передаются методу `build_and_run`, который вызывает метод `build` компонента. В отличие от запуска полного потока, зависимости-предшественники при этом не выполняются автоматически.

Просмотр вывода и журналов компонента

Чтобы просмотреть вывод и журналы отдельного компонента, нажмите **Инспектировать**.

Заморозка компонента

Заморозка компонента также замораживает все предшествующие компоненты. Используйте заморозку, если ожидается стабильный вывод от компонента *и всех его предшественников* и их нужно выполнить только один раз.

Заморозка предотвращает повторный запуск компонента и всех предшествующих компонентов и сохраняет последнее состояние их вывода. При последующих запусках потока используется сохранённый вывод.

Чтобы заморозить компонент, щёлкните по нему в рабочей области, открыв меню заголовка, и нажмите **Заморозить**.

24.2 Порты компонента

Вокруг границы каждого компонента находятся круглые значки портов. Они обозначают *точки подключения* компонента.

Порты принимают входные данные или генерируют выходные данные определённого типа. Тип данных можно определить по полю, к которому прикреплён порт, или по цвету порта. Например, поле **Системное сообщение** принимает данные типа `Message`, что обозначается синим значком порта.

При построении потоков соединяйте выходные порты с входными портами одного типа (цвета), чтобы передавать данные между компонентами. О программном представлении каждого типа данных см. в разделе **Типы данных Генеративный ИИ**.

- В рабочей области наведите курсор на порт, чтобы увидеть детали подключения. Щёлкните по порту, чтобы **найти** совместимые компоненты.
- Если два компонента используют несовместимые типы данных, воспользуйтесь компонентом-преобразователем, например **Конвертация типа**.

Динамические порты

Некоторые компоненты имеют порты, которые добавляются или удаляются динамически. Например, компонент **Шаблон промпта** принимает входные данные в фигурных скобках, и новые порты открываются при обнаружении значения в фигурных скобках в поле **Шаблон**.

Выбор типа вывода

Все компоненты генерируют вывод, который передаётся следующему компоненту или возвращается как финальный результат потока.

Некоторые компоненты могут генерировать несколько типов вывода:

- Если компонент генерирует все типы одновременно, в визуальном редакторе у него несколько выходных портов. В коде это представлено как `group_outputs=True`.

- Если компонент генерирует только один тип, нужно выбрать тип вывода, щёлкнув по метке вывода рядом с выходным портом. В коде это представлено как `group_outputs=False` или отсутствием параметра `group_outputs`.

Например, компонент языковой модели может выводить либо **Ответ модели**, либо **Языковую модель**. Вывод **Ответ модели** генерирует данные `Message` для передачи в порт `Message` другого компонента. Вывод **Языковая модель** должен быть подключён к компоненту с входом **Языковая модель**, например к **Структурированному выводу**, который использует привязанную LLM для обеспечения логики вывода.

Цвета портов

Цвет порта указывает тип данных, принимаемых или генерируемых портом. Например, порт **Сообщения** принимает или генерирует данные типа `Message`.

В следующей таблице перечислены типы данных и соответствующие цвета портов:

Тип данных	Цвет порта	Пример значка
<code>Data</code>	Красный	
<code>DataFrame</code>	Розовый	
<code>Embeddings</code>	Изумрудный	
<code>LanguageModel</code>	Фуксия	
<code>Memory</code>	Оранжевый	
<code>Message</code>	Индиго	
<code>Tool</code>	Голубой	
Неизвестные или множественные типы	Серый	

24.3 Код компонента

Вы можете редактировать компоненты в рабочем пространстве и в коде. Выберите компонент и нажмите **Код**, чтобы просмотреть и отредактировать базовый Python-код.

Код компонента в контексте создания и выполнения потока выполняет следующее:

- Определяет параметры конфигурации, отображаемые в визуальном редакторе.
- Проверяет входные данные на соответствие типам входов компонента.
- Обработывает данные, используя настроенные параметры, методы и функции.
- Передаёт результаты следующему компоненту в потоке.

Все компоненты наследуются от базового класса `Component`, определяющего интерфейс и поведение компонента. Например, компонент **Рекурсивный разделитель текста по символам** является дочерним классом `LCTextSplitterComponent`.

Код каждого компонента включает определения входов и выходов, отображаемых в рабочем пространстве как порты компонента. Например, у `RecursiveCharacterTextSplitter` четыре входа. Каждое определение входа указывает тип входных данных — например, `IntInput` — а также техническое имя, отображаемое имя, описание и другие параметры, которые определяют настройки в визуальном редакторе.

Кроме того, у компонентов есть методы, реализующие их функциональность. Например, у `RecursiveCharacterTextSplitter` два метода:

Метод `get_data_input` извлекает текст для разбиения из входных данных компонента, делая данные доступными классу. Метод `build_text_splitter` создаёт объект `RecursiveCharacterTextSplitter`, вызывая метод `build` родительского класса, после чего разбивает текст с помощью созданного сплиттера и передаёт результат следующему компоненту.

24.4 Версии компонентов

Версии компонентов и их состояния хранятся во внутренней базе данных Генеративный ИИ. При добавлении компонента в поток создаётся его отдельная копия на основе данных из базы. Эти копии не синхронизируются с обновлениями, которые могут появиться при обновлении версии Генеративный ИИ.

Иными словами, экземпляр компонента сохраняет номер версии и состояние с момента добавления в конкретный поток. Например, если компонент добавлен в версии 1.0, он остаётся в версии 1.0 *в этом потоке* до тех пор, пока вы явно не обновите его.

Обновление версий компонентов

При редактировании потока Генеративный ИИ уведомляет вас, если версия компонента в рабочей области устарела относительно базы данных:

- **Готово к обновлению:** обновление не содержит критических изменений.
- **Доступно обновление:** обновление может содержать критические изменения.

Критические изменения затрагивают входы и выходы компонента, что может привести к разрыву соединений и нарушению потока. После обновления может потребоваться изменить настройки компонента или заново подключить порты.

Доступны два способа обновления:

- Нажмите **Обновить**, чтобы обновить один компонент. Рекомендуется при отсутствии критических изменений.
- Нажмите **Обзор**, чтобы просмотреть все доступные обновления и создать снимок перед обновлением. Рекомендуется при наличии критических изменений.

Чтобы сохранить снимок потока перед обновлением, включите **Создать резервную копию потока перед обновлением**. Резервные потоки сохраняются в той же папке проекта с суффиксом (backup).

Чтобы обновить конкретные компоненты, выберите их и нажмите **Обновить компоненты**.

Компоненты обновляются до последней версии, доступной в используемой версии Генеративный ИИ.

24.5 Группировка компонентов

Несколько компонентов можно объединить в один для повторного использования. Это удобно для организации крупных потоков — например, объединения компонента **Агент RAG** со связанными инструментами или компонентами векторного хранилища.

1. Удерживая Shift, нажмите и перетащите, чтобы выделить все компоненты для объединения. Компоненты должны полностью находиться в выделенной области.
Альтернативно, удерживая Ctrl (Windows) или Cmd (Mac), щёлкайте по компонентам, чтобы добавлять их в группу по одному.
2. Отпустите кнопку мыши и клавишу, затем нажмите **Группировать**.

Сгруппированные компоненты управляются как единый компонент, включая имя, код и настройки.

Чтобы разгруппировать компоненты, щёлкните по группе в рабочей области, откройте меню заголовка, нажмите **Показать ещё**, затем выберите **Разгруппировать**.

Чтобы использовать группировку в других потоках, щёлкните по группе, откройте меню заголовка, нажмите **Показать ещё**, затем выберите **Сохранить** — компонент сохранится в меню **Основные компоненты** как пользовательский.

24.6 Устаревшие компоненты

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, однако рекомендуется как можно скорее заменить их поддерживаемыми аналогами. Рекомендуемые замены указаны в баннере **Устаревшее** на компонентах в ваших потоках, а также в примечаниях к выпуску и документации Генеративный ИИ.

Если вы не уверены, как заменить устаревший компонент, воспользуйтесь **поиском** по провайдеру, сервису или названию. Компонент мог быть заменён полностью новым компонентом, похожим аналогом или новой версией того же компонента в другой категории.

Если очевидной замены нет, рассмотрите возможность адаптации другого компонента под ваш сценарий. Например, многие **Основные компоненты** предоставляют общую функциональность с поддержкой нескольких провайдеров, например компонент **API-запрос**.

Если ни один из вариантов не подходит, вы можете использовать код устаревшего компонента для создания пользовательского компонента или открыть обсуждение.

Чтобы устаревшие компоненты не предлагались в новых потоках, они скрыты по умолчанию. В визуальном редакторе нажмите **Настройки компонентов**, чтобы переключить фильтр **Устаревшее**.

25. Основные компоненты

25.1 Ввод и вывод

25.1.1 Ввод в чат и Вывод

Компоненты **Ввод в чат** и **Вывод чата** необходимы для взаимодействия с потоком в **Playground**. Подробнее см. в разделе Тестирование потоков в Playground. Компоненты **Ввод в чат** и **Вывод чата** предназначены для обработки диалоговых взаимодействий в модуле «Генеративный ИИ».

Ввод в чат

Компонент **Ввод в чат** принимает текст и файлы — например, сообщение чата или прикрепленный файл. Эти данные передаются другим компонентам в виде объектов Message, содержащих предоставленный ввод, а также связанные метаданные: отправителя, идентификатор сессии, временную метку и вложения.

Начальный ввод *не должен* предоставляться в виде готового объекта Message — компонент **Ввод в чат** создаёт его самостоятельно и передаёт другим компонентам в потоке.

Параметры компонента «Ввод в чат»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
input_value	Ввод текста	Входной параметр. Текстовая строка сообщения, передаваемая в качестве ввода.
sender	Тип отправителя	Входной параметр. Идентифицирует отправителя как User или Language Model.
sender_name	Имя отправителя	Входной параметр. Имя отправителя. Если не указано, используется User или Language Model.
session_id	Идентификатор сессии	Входной параметр. Уникальный идентификатор чата. Если не указан, используется текущий идентификатор сессии.
files	Файлы	Входной параметр. Файлы, отправляемые вместе с сообщением.
background_color	Цвет фона	Входной параметр. Цвет фона иконки.

Название	Отображаемое название	Описание
chat_icon	Иконка	Входной параметр. Иконка сообщения.
should_store_message	Сохранять сообщения	Входной параметр. Определяет, следует ли сохранять сообщение в истории чата.
text_color	Цвет текста	Входной параметр. Цвет текста имени отправителя.

Подробнее об объекте Message, включая параметры ввода, напрямую сопоставленные с его атрибутами, см. в разделе Типы данных Message.

Метод создания сообщений для компонента «Ввод в чат»

Класс ChatInput предоставляет асинхронный метод для создания и сохранения объекта Message на основе входных параметров. Объект Message создаётся в методе message_response класса ChatInput с помощью фабричного метода Message.create().

```
message = await Message.create(
    text=self.input_value,
    sender=self.sender,
    sender_name=self.sender_name,
    session_id=self.session_id,
    files=self.files,
    properties={
        "background_color": background_color,
        "text_color": text_color,
        "icon": icon,
    },
)
```

Вывод чата

Компонент **Вывод чата** принимает данные типов Message, Data или DataFrame от других компонентов, при необходимости преобразует их в объект Message и возвращает финальный результат в виде сообщения чата. Подробнее о типах данных см. в разделе Типы данных Генеративный ИИ.

В **Playground** вывод чата ограничен частями объекта Message, релевантными для интерфейса чата: текстом ответа, именем отправителя и вложениями. Чтобы просмотреть метаданные сообщения, откройте журналы сообщений в **Playground**.

При использовании API Генеративный ИИ ответ содержит объект Message компонента **Вывод чата** вместе с другими данными выполнения потока. Ответы API могут быть очень подробными, поэтому в приложениях следует предусматривать код для извлечения нужных данных и возврата их пользователю. Пример см. в разделе Быстрый старт Генеративный ИИ.

Параметры компонента «Вывод чата»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
input_value	Входные данные	Входной параметр. Текстовая строка сообщения, передаваемая в качестве вывода.

Название	Отображаемое название	Описание
should_store_message	Сохранять сообщения	Входной параметр. Определяет, следует ли сохранять сообщение в истории чата.
sender	Тип отправителя	Входной параметр. Идентифицирует отправителя как User или Language Model.
sender_name	Имя отправителя	Входной параметр. Имя отправителя. Если не указано, используется User или Language Model.
session_id	Идентификатор сессии	Входной параметр. Уникальный идентификатор чата. Если не указан, используется текущий идентификатор сессии.
data_template	Шаблон данных	Входной параметр. Шаблон для преобразования входных данных Data в текст. Если не указан, динамически задаётся ключом text объекта Data.
background_color	Цвет фона	Входной параметр. Цвет фона иконки.
chat_icon	Иконка	Входной параметр. Иконка сообщения.
text_color	Цвет текста	Входной параметр. Цвет текста имени отправителя.
clean_data	Очистка данных	Входной параметр. При включении входные данные DataFrame очищаются при преобразовании в текст: удаляются пустые строки, пустые ячейки и множественные переносы строк.

Подробнее об объекте Message, включая входные параметры, напрямую сопоставленные с его атрибутами, см. в разделе Типы данных Message.

Использование компонентов «Ввод в чат» и «Вывод чата» в потоке

Чтобы использовать компоненты **Ввод в чат** и **Вывод чата** в потоке, подключите их к компонентам, которые принимают или генерируют данные Message.

Например, следующий поток соединяет **Ввод в чат** и **Вывод чата** с компонентом **Языковая модель**, создавая простой чат-поток на основе LLM.

Подробные примеры использования компонентов **Ввод в чат** и **Вывод чата** в потоках:

- Быстрый старт Генеративный ИИ: создайте и запустите базовый агентный поток.
- Шаблон **Основной промпт**: создайте чат-поток на основе LLM, принимающий сообщение чата и промпт с дополнительными инструкциями для LLM. Многие другие шаблоны Генеративный ИИ также используют компоненты **Ввод в чат** и **Вывод чата**.
- Подключение приложения к агентам: изучите продвинутые концепции агентных потоков и промптов, включая запуск потоков из внешних приложений.

Отправка ввода чата через API Генеративный ИИ

Вы можете использовать API Генеративный ИИ для запуска потока, отправляя входные данные в компонент **Ввод в чат**:

```
curl --request POST \
  --url "http://$ASTRA_SERVER_ADDRESS/api/v1/run/$FLOW_ID" \
  --header "Content-Type: application/json" \
  --header "x-api-key: $ASTRA_API_KEY" \
  --data '{
    "input_value": "What\'s the recommended way to install Docker on Mac M1?",
    "output_type": "chat",
    "input_type": "chat"
  }'
```

При запуске потоков через API полезная нагрузка должна содержать значения входных параметров компонента **Ввод в чат**, например `input_value`.

Указывать все параметры необязательно. Например, если `session_id` не указан, используется идентификатор сессии по умолчанию. Чтобы использовать пользовательский идентификатор сессии, включите `session_id` в запрос:

```
curl --request POST \
  --url "http://$ASTRA_SERVER_ADDRESS/api/v1/run/$FLOW_ID" \
  --header "Content-Type: application/json" \
  --header "x-api-key: $ASTRA_API_KEY" \
  --data '{
    "input_value": "What\'s the recommended way to install Docker on Mac M1?",
    "session_id": "$USER_ID",
    "output_type": "chat",
    "input_type": "chat"
  }'
```

Подробнее см. в разделе **Запуск потоков через API Генеративный ИИ**.

25.1.2 Ввод текста и Вывод текста

Компоненты **Ввод текста** и **Вывод текста** не поддерживаются в **Playground**. Поскольку данные не отформатированы как сообщение чата, они не отображаются в **Playground** и не позволяют вести диалог с потоком.

Если вы хотите вести диалог с потоком в **Playground**, используйте компоненты **Ввод в чат** и **Вывод чата**.

Компоненты **Ввод текста** и **Вывод текста** предназначены для потоков, которые принимают или генерируют простые текстовые строки. Они не поддерживают полноценные диалоговые взаимодействия.

Передача метаданных в формате чата в компонент **Ввод текста** или **Вывод текста** не изменяет поведение компонента — результатом по-прежнему будет обычная текстовая строка.

Ввод текста

Компонент **Ввод текста** принимает текстовую строку и передаёт её другим компонентам в виде объекта Message, содержащего только переданную строку в атрибуте text.

Компонент принимает только параметр **Текст** (input_value) — строку, которую можно ввести непосредственно в компонент или получить как данные типа Message от других компонентов.

Входные данные *не должны* передаваться в виде полного объекта Message: компонент **Ввод текста** самостоятельно формирует объект Message, который затем передаётся другим компонентам в потоке.

Вывод текста

Компонент **Вывод текста** принимает данные типа Message от других компонентов и возвращает только атрибут text в виде упрощённого объекта Message.

Компонент принимает только параметр **Текст** (input_value) — строку, которую можно ввести непосредственно в компонент или получить как данные типа Message от других компонентов.

25.1.3 Вебхук

Компонент **Вебхук** определяет триггер, который запускает поток при получении HTTP POST-запроса.

Запуск вебхука

Когда вы добавляете компонент **Вебхук** в поток, в панели **Доступ к API** появляется вкладка **Вебхук curl**. Она автоматически генерирует фрагмент кода с HTTP POST-запросом для запуска потока через компонент **Вебхук**. Например:

```
curl -X POST
"http://$ASTRA_SERVER_ADDRESS/api/v1/webhook/$FLOW_ID"
-H 'Content-Type: application/json'
-H 'x-api-key: $ASTRA_API_KEY'
-d '{"any": "data"}'
```

Подробнее см. в разделе [Запуск потоков с помощью вебхуков](#).

Параметры вебхука

Название	Отображаемое название	Описание
data	Полезная нагрузка	Входной параметр. Принимает полезную нагрузку от внешних систем через HTTP POST-запросы.
curl	curl	Входной параметр. Шаблон команды curl для выполнения запросов к этому вебхуку.
endpoint	Конечная точка	Входной параметр. URL конечной точки, по которому вебхук принимает запросы.
output_data	Данные	Выходной параметр. Обработанные данные из входа вебхука. Возвращает пустой объект Data, если входные данные не предоставлены. Если

Название	Отображаемое название	Описание
		входные данные не являются корректным JSON, компонент Вебхук оборачивает их в объект <code>payload</code> , чтобы они могли использоваться как входные данные для запуска потока.

25.2 Обработка данных

25.2.1 Операции с данными

Компонент **Операции с данными** выполняет операции над объектами `Data`, включая извлечение, фильтрацию и редактирование ключей и значений в `Data`. Для всех вариантов см. [Доступные операции с данными](#). Выводом является новый объект `Data`, содержащий изменённые данные после выполнения выбранной операции.

Использование компонента **Операции с данными** в потоке

Следующий пример демонстрирует, как использовать компонент **Операции с данными** в потоке, используя данные из полезной нагрузки вебхука:

1. Создайте поток с компонентом **Вебхук** и компонентом **Операции с данными**, затем подключите вывод компонента **Вебхук** к входу **Данные** компонента **Операции с данными**.

Все операции в компоненте **Операции с данными** требуют как минимум один ввод `Data` от другого компонента. Если предыдущий компонент не генерирует вывод `Data`, вы можете использовать другой компонент, например компонент **Конвертировать Тип**, чтобы переформить данные перед передачей их в компонент **Операции с данными**. В качестве альтернативы можно рассмотреть использование компонента, предназначенного для обработки исходного типа данных, например компонент **Парсер** или компонент **Операции с DataFrame**.

2. В поле **Операции** выберите операцию, которую хотите выполнить над входящими `Data`. Для этого примера выберите операцию **Выберите ключи**.

Можно выбрать только одну операцию. Если необходимо выполнить несколько операций над данными, вы можете соединить несколько компонентов **Операции с данными** последовательно, чтобы выполнить каждую операцию по очереди. Для более сложных многошаговых операций рассмотрите возможность использования компонента, например компонент **Умное преобразование**.

3. В разделе **Выберите ключи** добавьте ключи для `name`, `username` и `email`.

Нажмите **Add more**, чтобы добавить поле для каждого ключа.

Для этого примера предположим, что вебхук будет получать последовательные полезные нагрузки, которые всегда содержат ключи `name`, `username` и `email`. Операция **Выберите ключи** извлекает значение этих ключей из каждой входящей полезной нагрузки.

3. Опционально: если вы хотите просмотреть вывод в **Playground**, подключите вывод компонента **Операции с данными** к компоненту **Вывод чата**.

4. Чтобы протестировать поток, отправьте следующий запрос на endpoint вебхука вашего потока. Для получения дополнительной информации об endpoint вебхука см. Trigger flows with webhooks.

```
curl -X POST "http://$Astra_Multiagent_AI_SERVER_URL/api/v1/webhook/$FLOW_ID" \
-H "Content-Type: application/json" \
-H "x-api-key: $Astra_Multiagent_AI_API_KEY" \
-d '{
  "id": 1,
  "name": "Leanne Graham",
  "username": "Bret",
  "email": "Sincere@april.biz",
  "address": {
    "street": "Main Street",
    "suite": "Apt. 556",
    "city": "Springfield",
    "zipcode": "92998-3874",
    "geo": {
      "lat": "-37.3159",
      "lng": "81.1496"
    }
  },
  "phone": "1-770-736-8031 x56442",
  "website": "hildegard.org",
  "company": {
    "name": "Acme-Corp",
    "catchPhrase": "Multi-layered client-server neural-net",
    "bs": "harness real-time e-markets"
  }
}'
```

6. Чтобы просмотреть Data, полученный в результате операции **Выберите ключи**, выполните одно из следующих действий:

- Если вы подключили компонент **Вывод чата**, откройте **Playground**, чтобы увидеть результат в виде сообщения чата.
- Нажмите **Просмотреть вывод** на компоненте **Операции с данными**.

Параметры операции с данными

Многие параметры зависят от выбранного **Операция** (operation).

Название	Отображать Название	Информация
data	Данные	Входной параметр. Объект Data, над которым будет выполнена операция.
operation	Операция	Входной параметр. Операция, которую следует выполнить над данными. См. Доступные операции с данными
select_keys_input	Выберите ключи	Входной параметр. Список ключей для выбора из данных.
filter_key	Ключ фильтра	Входной параметр. Ключ, по которому будет выполнена фильтрация.

Название	Отображать Название	Информация
operator	Оператор сравнения	Входной параметр. Оператор, применяемый для сравнения значений.
filter_values	Значения фильтра	Входной параметр. Список значений для фильтрации.
append_update_data	Добавить или обновить	Входной параметр. Данные, которые следует добавить или обновить в существующих данных.
remove_keys_input	Удалить ключи	Входной параметр. Список ключей для удаления из данных.
rename_keys_input	Переименование ключей	Входной параметр. Список ключей для переименования в данных.
mapped_json_display	JSON на карту	Входной параметр. JSON-структура для исследования при выборе пути. Применяется только к операции Путь Selection . Для получения дополнительной информации см. Примеры использования операции выбора пути.
selected_key	Выберите путь	Входной параметр. Выражение пути JSON для извлечения значений. Применяется только к операции Путь Selection . Для получения дополнительной информации см. Примеры использования операции выбора пути.
query	JQ-выражение	Входной параметр. Выражение jq для продвинутой фильтрации и трансформации JSON. Применяется только к операции JQ-выражение . Для получения дополнительной

Название	Отображать Название	Информация информации см. Примеры JQ-выражений.

Доступные операции с данными

Опции для входного параметра operations перечислены ниже. Все операции работают с входным объектом Data.

Название	Требуется	Входные	Процесс
Выберите ключи		select_keys_input	Выбирает определённые ключи из данных.
Буквальное вычисление	None		Оценивает строковые значения как литералы Python.
Объединить	None		Объединяет несколько объектов данных в один.
Значения фильтра	filter_key, filter_values, operator		Фильтрует данные по паре ключ-значение.
Добавить или обновить	append_update_data		Добавляет или обновляет пары ключ-значение.
Удалить ключи	remove_keys_input		Удаляет указанные ключи из данных.
Переименование ключей	rename_keys_input		Переименовывает ключи в данных.
Выбор пути	mapped_json_display, selected_key		Извлекает значения из вложенных JSON-структур с помощью выражений пути.
JQ-выражение	query		Выполняет продвинутые запросы JSON с использованием синтаксиса jq для фильтрации, проекций и трансформаций.

Примеры использования операции выбора пути

Используйте операцию выбора Путь для извлечения значений из вложенных JSON-структур с путями в точечной нотации.

1. В раскрывающемся списке **Операции** выберите **Выбор пути**.
2. В поле **JSON на карту** введите вашу JSON-структуру.

В этом примере используется следующая JSON-структура.

```
{
  "user": {
    "profile": {
      "name": "John Doe",
      "email": "john@example.com"
    },
    "settings": {
      "theme": "dark"
    }
  }
}
```

```
}  
}  
}
```

Выпадающий список **Выберите путь** автоматически заполняется доступными путями. 3. В выпадающем списке **Выбрать пути** выберите путь. Вы можете выбрать пути, такие как `.user.profile.name`, чтобы извлечь “John Doe”, или выбрать `.user.settings.theme`, чтобы извлечь “dark”.

Примеры JQ-выражений

Используйте операцию **JQ Expressions**, чтобы использовать язык запросов jq для более продвинутой фильтрации JSON.

1. В выпадающем списке **Операции** выберите **JQ-выражение**.
2. В поле **JQ-выражение** введите фильтр jq, чтобы выполнить запрос к входу Данные компонента **Операции с данными**.

Для этой примерной структуры JSON введите выражения, такие как `.user.profile.name`, чтобы извлечь “John Doe”, `.user.profile | {name, email}`, чтобы проецировать поля в новый объект, или `.user.profile | tostring`, чтобы преобразовать поле в строку.

```
{  
  "user": {  
    "profile": {  
      "name": "John Doe",  
      "email": "john@example.com"  
    },  
    "settings": {  
      "theme": "dark"  
    }  
  }  
}
```

25.2.2 Операции с DataFrame

Компонент **Операции с DataFrame** выполняет операции над строками и столбцами DataFrame (таблица), включая изменения схемы, изменения записей, сортировку и фильтрацию. Для всех вариантов см. Операции с параметрами DataFrame.

Выходом является новый DataFrame, содержащий изменённые данные после выполнения выбранной операции.

Использование компонента Операции с DataFrame в потоке

Следующие шаги объясняют, как настроить компонент **Операции с DataFrame** в потоке. Вы можете следовать примеру или использовать свой собственный поток. Единственное требование — предшествующий компонент должен создавать вывод DataFrame, который можно передать в компонент **Операции с DataFrame**.

1. Создайте новый поток или используйте существующий поток.

Пример: поток извлечения данных из API-ответа

Следующий пример использует пять компонентов для извлечения данных Данные из ответа API, преобразования его в DataFrame и последующей обработки табличных данных с помощью компонента **Операции с DataFrame**. Шестой компонент, **Вывод чата**, является необязательным в этом примере. Он служит лишь удобным способом просмотреть окончательный вывод в **Песочница**, вместо того чтобы проверять журналы компонента.

Если вы хотите использовать этот пример для тестирования компонента **Операции с DataFrame**, выполните следующее:

1. Создайте поток со следующими компонентами:
 - API-запрос

- **Языковая модель**
 - **Умное преобразование**
 - **Конвертировать Тип**
2. Настройте компонент **Умное преобразование** и его зависимости:
 - **API-запрос:** Настройте компонент **API-запрос** для получения JSON-данных с выбранного вами конечного пункта, а затем подключите вывод **API Ответ** к входу **Данные** компонента **Умное преобразование**.
 - **Языковая модель:** Выберите предпочтительного провайдера и модель, затем введите действительный API-ключ. Измените вывод на **Языковая модель**, а затем подключите вывод **Языковая модель** к входу **Языковая модель** компонента **Умное преобразование**.
 - **Умное преобразование:** В поле **Инструкции** введите инструкции на естественном языке для извлечения данных из ответа API. Ваши инструкции зависят от содержимого ответа и желаемого результата. Например, если ответ содержит большое поле `result`, вы можете дать инструкции вроде `извлеките данные из поля result` и преобразуйте их в объект `Data`.
 3. Преобразуйте вывод **Данные** компонента **Умное преобразование** в `DataFrame`:
 1. Подключите вывод **Отфильтрованные данные** к входу **Данные** компонента **Конвертировать Тип**.
 2. Установите **Тип выхода** компонента **Конвертировать Тип** в значение **DataFrame**.

Теперь поток готов к добавлению компонента **Операции с DataFrame**.

2. Добавьте компонент **Операции с DataFrame** в поток, а затем подключите вывод `DataFrame` из другого компонента к входу **DataFrame**.

Все операции в компоненте **Операции с DataFrame** требуют как минимум один ввод `DataFrame` из другого компонента. Если компонент не генерирует вывод `DataFrame`, вы можете использовать другой компонент, например компонент **Конвертировать Тип**, чтобы переформатировать данные перед передачей их в компонент **Операции с DataFrame**. Кроме того, можно рассмотреть использование компонента, предназначенного для обработки исходного типа данных, например компонент **Парсер** или компонент **Операции с данными**.

Если вы следуете примеру, подключите порт **DataFrame Вывод** компонента **Конвертировать Тип** к входу **DataFrame**.

3. В поле **Операции** выберите операцию, которую хотите выполнить над входящим `DataFrame`. Например, операция **Фильтрация** фильтрует строки по указанному столбцу и значению.

Вы можете выбрать только одну операцию. Если необходимо выполнить несколько операций над данными, вы можете соединить несколько компонентов **Операции с DataFrame** последовательно, чтобы выполнить каждую операцию по очереди. Для более сложных многошаговых операций, таких как кардинальные изменения схемы или повороты, рассмотрите использование компонента, управляемого LLM, например компонент **Структурированный вывод** или компонент **Умное преобразование**, в качестве замены или подготовки для компонента **Операции с DataFrame**.

Если вы следуете примеру потока, выберите любую операцию, которую хотите применить к данным, извлечённым компонентом **Умное преобразование**. Чтобы просмотреть

содержимое входного DataFrame, нажмите **Запустить компонент** на компоненте **Конвертировать Тип**, а затем **Исследовать вывод**. Если DataFrame выглядит некорректным, нажмите **Исследовать вывод** на каждом предыдущем компоненте, чтобы определить, где возникла ошибка, и при необходимости измените конфигурацию потока. Например, если компонент **Умное преобразование** не извлек ожидаемые поля, измените инструкции или проверьте, что указанные поля присутствуют в выводе **API Ответ**.

4. Настройте параметры операции. Конкретные параметры зависят от выбранной операции. Например, если вы выбираете операцию **Фильтрация**, необходимо задать условие фильтрации, используя параметры **Имя столбца**, **Значение фильтра** и **Оператор фильтра**. Для получения дополнительной информации см. **Операции с параметрами DataFrame**
5. Чтобы протестировать поток, нажмите **Запустить компонент** в компоненте **Операции с DataFrame**, а затем нажмите **Исследовать вывод**, чтобы увидеть новый DataFrame, созданный операцией **Фильтрация**.

Если вы хотите просмотреть вывод в **Песочница**, подключите вывод компонента **Операции с DataFrame** к компоненту **Вывод чата**, повторно запустите компонент **Операции с DataFrame**, а затем нажмите **Песочница**.

Для другого примера см. Conditional looping.

Операции с параметрами DataFrame

Большинство параметров **Операции с DataFrame** являются условными, поскольку они применимы только к определённым операциям.

Единственными постоянными параметрами являются **DataFrame (df)**, который является входом DataFrame, и **Операция (operation)**, который определяет операцию, выполняемую над DataFrame. После выбора операции условные параметры для этой операции появляются в компоненте **Операции с DataFrame**.

- Добавить столбец
- Удалить столбец
- Фильтр
- Первая часть
- Переименовать столбец
- Заменить Уникальный ID
- Выбрать столбцы
- Сортировать
- Последние строки
- Удалить дубликаты

Операция **Add Column** позволяет добавить новый столбец в DataFrame с постоянным значением.

Параметры: **Новое имя столбца** (new_column_name) и **Новое значение столбца** (new_column_value).

Операция **Drop Column** позволяет удалить столбец из DataFrame, указанный через **Имя столбца** (column_name).

Операция ****Фильтрация**** позволяет отфильтровать DataFrame по заданному условию. Выходом является DataFrame, содержащий только строки, соответствующие условию фильтра.

Укажите следующие параметры: - **Имя столбца** (column_name): Имя столбца, по которому выполнять фильтрацию. - **Значение фильтра** (filter_value): Значение, по которому фильтровать. - **Оператор фильтра** (filter_operator): Оператор, используемый для фильтрации; один из equals (по умолчанию), not equals, contains, not contains, starts with, ends with, greater than или less than.

Операция **Head** позволяет получить первые n строк DataFrame, где n задаётся в **Количество строк** (num_rows). По умолчанию — 5. Выходом является DataFrame, содержащий только выбранные строки.

Операция **Rename Column** позволяет переименовать существующий столбец в DataFrame. Параметры: **Имя столбца** (column_name) — текущее имя, и **Новое имя столбца** (new_column_name).

Операция **Replace Value** позволяет заменить значения в определённом столбце DataFrame. Эта операция заменяет целевое значение новым значением. Все ячейки, соответствующие целевому значению, заменяются новым значением в выходном DataFrame.

Укажите следующие параметры: - **Имя столбца** (column_name): Имя столбца, который нужно изменить. - **Заменяемое значение** (replace_value): Значение, которое необходимо заменить. - **Значение для замены** (replacement_value): Новое значение.

Операция **Select Columns** позволяет выбрать один или несколько конкретных столбцов из DataFrame. Укажите список имён столбцов в **Столбцы для выбора** (columns_to_select). В визуальном редакторе нажмите **Add More**, чтобы добавить несколько полей, и введите имя столбца в каждое поле. Выходом является DataFrame, содержащий только указанные столбцы.

Операция **Sort** позволяет отсортировать DataFrame по определённому столбцу в порядке возрастания или убывания.

Укажите следующие параметры: - **Имя столбца** (column_name): Имя столбца, по которому выполнять сортировку. - **Сортировать по возрастанию** (ascending): Определяет, сортировать в порядке возрастания или убывания. Если включено (true), сортировка по возрастанию; если отключено (false), сортировка по убыванию. По умолчанию: включено (true).

Операция **Tail** позволяет получить последние n строк DataFrame, где n задаётся в **Количество строк** (num_rows). По умолчанию — 5. Выходом является DataFrame, содержащий только выбранные строки.

Операция **Drop Duplicates** удаляет строки из DataFrame, определяя все дублирующиеся значения в одном столбце. Единственный параметр — **Имя столбца** (column_name).

Когда поток запускается, все строки с дублирующимися значениями в указанном столбце удаляются. Вывод представляет собой DataFrame, содержащий все столбцы из оригинального DataFrame, но только строки с недублирующимися значениями.

25.2.3 Динамическое создание данных

Компонент **Динамическое создание данных** создаёт объект Data или Message с настраиваемыми полями. Определите таблицу в поле **Конфигурация входа**, и компонент создаст соответствующие входные и выходные соединения.

Использование компонента в потоке

Следующий пример демонстрирует, как использовать компонент **Динамическое создание данных** для создания структурированного объекта Data или Message из нескольких источников.

1. Добавьте компонент **Динамическое создание данных** в ваш поток.

2. Чтобы определить поля ваших данных, в поле **Конфигурация входа** нажмите **Открыть таблицу**.

3. Чтобы добавить строки в таблицу, нажмите **Добавить новую строку**.

Добавление новой строки создаёт входные и выходные соединения для **Field Type**. Например, если вы добавите поле типа Text, то входные и выходные соединения Text будут добавлены к компоненту.

Для каждой новой строки настройте **Field Name** и **Field Type**:

- **Field Name** — имя поля, используемое как внутренний ключ и отображаемая метка.
- **Field Type** — тип создаваемого входного поля. Доступные варианты:
 - **Text** — принимает прямой текстовый ввод или вывод Text / Message от других компонентов.
 - **Data** — принимает ввод Data от других компонентов.
 - **Number** — принимает прямой числовой ввод или вывод Text / Message от других компонентов.
 - **Handle** — принимает вывод Text, Data или Message от других компонентов.
 - **Boolean** — принимает логические значения. Не может принимать ввод от другого компонента.

Для получения дополнительной информации см. Типы данных Генеративный ИИ.

4. В зависимости от выбранных **Field Type** подключите вывод от других компонентов для динамического заполнения входов или вручную введите значения в поля компонента.

5. Выберите желаемый тип вывода на выходном порте компонента. Компонент выводит либо объект Data, содержащий все значения полей, либо Message, содержащий все значения полей, отформатированные как текстовая строка.

Параметры компонента

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
form_fields	Конфигурация входа	Входной параметр. Таблица, определяющая динамические поля формы.
include_metadata	Включать метаданные	Входной параметр. Определяет, включать ли метаданные конфигурации формы в вывод.
form_data	Данные	Выходной параметр. Объект Data, содержащий все значения полей из динамических входов.
message	Сообщение	Выходной параметр. Объект Message, содержащий все значения

Название	Отображаемое название	Описание полей, отформатированные как текстовая строка.

25.2.4 Парсер

Компонент **Парсер** извлекает текст из структурированных данных (DataFrame или Data) с использованием шаблона или прямой сериализации. Вывод представляет собой объект Message, содержащий разобранный текст.

Это универсальный компонент для извлечения и обработки данных в ваших потоках. Примеры использования компонента **Парсер** в потоках:

- Пример компонента **Пакетный запуск**
- Пример компонента **Структурированный вывод**
- Шаблон **Financial Report Parser**
- Запуск потоков с помощью вебхуков
- Создание чат-бота RAG с векторным поиском

Режимы разбора

Компонент **Парсер** работает в двух режимах: **Парсер** и **Stringify**.

В режиме **Парсер** вы создаёте шаблон для текстового вывода, который может содержать буквальныe строки и переменные для извлечённых ключей.

Используйте фигурные скобки для определения переменных в любом месте шаблона. Переменные должны соответствовать ключам во входных данных DataFrame или Data, например именам столбцов. Например, {name} извлекает значение ключа name. Для получения дополнительной информации о содержимом и структуре объектов DataFrame и Data см. Типы данных Генеративный ИИ.

Если шаблон содержит как буквальныe текст, так и переменные, используйте двойные фигурные скобки для экранирования буквальныx скобок, чтобы они не интерпретировались как переменные. Например: This is a template with {{literal text in curly braces}} and a {variable}.

Если шаблон содержит множество буквальныx фигурных скобок, например JSON-структуры, рассмотрите использование шаблонов Mustache. Для получения дополнительной информации см. Использование шаблонов Mustache в шаблонах промптов.

При выполнении потока компонент **Парсер** перебирает входные данные, создавая объект Message для каждого разобранныx элемента. Например, разбор DataFrame создаёт отдельный Message для каждой строки, заполненный уникальными значениями из этой строки.

Шаблон резюме сотрудника

Этот пример шаблона извлекает данные о сотруднике в текстовое описание с датой найма и текущей должностью:

```
{employee_first_name} {employee_last_name} was hired on {start_date}.
Their current position is {job_title} ({grade}).
```

Полученный объект Message заменяет переменные соответствующими извлечёнными значениями.

При выполнении потока компонент **Парсер** перебирает каждую строку DataFrame, подставляя значения из соответствующих полей. Результирующий текст для каждой строки возвращается как объект Message.

Ниже перечислены параметры режима **Парсер**.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
input_data	Данные или DataFrame	Входной параметр. Входные данные Data или DataFrame для разбора.
pattern	Шаблон	Входной параметр. Шаблон форматирования, использующий обычный текст и переменные для ключей ({KEY_NAME}). См. примеры выше.
sep	Разделитель	Входной параметр. Строка-разделитель для строк или элементов. По умолчанию: <code>\n</code> (новая строка).
clean_data	Очистить данные	Определяет, следует ли удалять пустые строки и элементы в каждой ячейке или ключе входных данных DataFrame или Data. По умолчанию: true.

Используйте режим **Stringify**, чтобы преобразовать весь ввод напрямую в текст без использования шаблонов или выбора ключей.

Ниже перечислены параметры режима **Stringify**.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
input_data	Данные или DataFrame	Входной параметр. Входные данные Data или DataFrame для разбора.
sep	Разделитель	Входной параметр. Строка-разделитель для строк или элементов. По умолчанию: <code>\n</code> (новая строка).
clean_data	Очистить данные	Определяет, следует ли удалять пустые строки и элементы в каждой ячейке или ключе входных данных DataFrame или Data. По умолчанию: true.

Тестирование и устранение неполадок

Чтобы протестировать компонент **Парсер**, нажмите **Запустить компонент**, а затем **Просмотреть вывод**, чтобы увидеть объект Message с разобранным текстом. Также можно подключить компонент **Вывод чата**, чтобы просмотреть результат в **Playground**.

Если вывод Message содержит пустые или неожиданные значения, причиной может быть несоответствие между вводом и режимом разбора, наличие пустых значений во вводе или несовместимость ввода с извлечением обычного текста.

Например, при использовании следующего шаблона для разбора DataFrame:

```
{employee_first_name} {employee_last_name} is a {job_title} ({grade}).
```

Если `employee_first_name` пустой, а `grade` равен `null`, результат может выглядеть так:
Smith is a Software Engineer (null).

Чтобы устранить отсутствующие или неожиданные значения:

- Убедитесь, что переменные в шаблоне соответствуют ключам во входных данных Data или DataFrame. Чтобы увидеть данные, поступающие в компонент **Парсер**, нажмите **Просмотреть вывод** на компоненте, который отправляет эти данные.
- Проверьте исходные данные на наличие отсутствующих или некорректных значений. Существует несколько способов устранить несоответствия:
 - Исправьте исходные данные напрямую.
 - Используйте другие компоненты для исправления или фильтрации аномалий перед передачей данных в **Парсер**, например **Операции с данными**, **Структурированный вывод** или **Умное преобразование**.
- Включите параметр **Очистить данные** компонента **Парсер**, чтобы пропускать пустые строки.

25.2.5 Разделить текст

Компонент **Разделить текст** разбивает данные на фрагменты (чанки) на основе параметров, таких как размер фрагмента и разделитель. Он часто используется для разбиения данных, которые затем токенизируются и встраиваются в векторные базы данных. Примеры использования см. в разделах Использование компонентов модели эмбедингов в потоке и Создать чат-бот с векторным RAG.

Компонент принимает Message, Data или DataFrame и возвращает данные в виде **Chunks** либо **DataFrame**. Вывод **Chunks** возвращает список объектов Data, содержащих отдельные текстовые фрагменты. Вывод **DataFrame** возвращает список фрагментов в виде структурированного DataFrame с дополнительными столбцами text и metadata.

Параметры компонента

Параметры компонента **Разделить текст** управляют тем, как текст разбивается на фрагменты, в частности `chunk_size`, `chunk_overlap` и `separator`.

Чтобы протестировать поведение разбиения, добавьте компонент **Ввод текста** или **Чтение файла** с примерными данными, нажмите **Run component** на компоненте **Разделить текст**, а затем нажмите **Inspect output**, чтобы просмотреть список фрагментов и их метаданные. Столбец **text** содержит фактические текстовые фрагменты, созданные согласно вашим настройкам разбиения. Если фрагменты не разбиваются так, как ожидается, скорректируйте параметры, повторно запустите компонент и проверьте новый вывод.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
data_inputs	Ввод	Входной параметр. Данные для разбиения. Должны быть в формате

Название	Отображаемое название	Описание
		Message, Data или DataFrame.
chunk_overlap	Перекрытие фрагментов	Входной параметр. Количество символов, перекрывающихся между соседними фрагментами. Это помогает сохранять контекст на границах фрагментов. При встрече разделителя перекрытие применяется в точке разбиения: последующий фрагмент содержит последние <i>n</i> символов предыдущего. По умолчанию: 200.
chunk_size	Размер фрагмента	Входной параметр. Целевая длина каждого фрагмента после разбиения. Данные сначала разбиваются по разделителю, затем фрагменты меньше chunk_size объединяются до этого предела. Однако если начальное разбиение по разделителю даёт фрагменты, превышающие chunk_size, они не делятся дальше и не объединяются с меньшими — такие фрагменты выводятся как есть. По умолчанию: 1000. См. Ошибки токенизации из-за размера фрагмента.
separator	Разделитель	Входной параметр. Строка, определяющая символ разбиения, например \n для разбиения по символам новой строки, \n\n для разбиения по абзацам или }, для разбиения в конце JSON-объектов. Можно указать

Название	Отображаемое название	Описание
		строку напрямую или передать её из другого компонента как данные Message.
text_key	Текстовый ключ	Входной параметр. Ключ, используемый для столбца текста, который извлекается из входных данных и затем разбивается. По умолчанию: text.
keep_separator	Сохранить разделитель	Входной параметр. Определяет, как обрабатывать разделители в выходных фрагментах. Варианты: False (удалить разделители), True (сохранять разделители без предпочтения места), Start (размещать разделители в начале фрагментов), End (размещать разделители в конце фрагментов). По умолчанию: False.

Ошибки токенизации из-за размера фрагмента

При использовании компонента **Разделить текст** совместно с моделями эмбедингов (особенно моделями NVIDIA, такими как `nvidia/nv-embed-v1`) может потребоваться уменьшить размер фрагмента до 500 и менее, даже если модель формально поддерживает большее ограничение по токенам. Компонент **Разделить текст** не всегда строго соблюдает указанный размер фрагмента, и отдельные фрагменты могут превышать заданный предел. Если вы сталкиваетесь с ошибками токенизации, измените стратегию разбиения: уменьшите размер фрагмента, измените длину перекрытия или используйте более частый разделитель. После изменений проверьте конфигурацию, запустив поток и проверив вывод компонента.

Другие варианты разбиения текста

См. компоненты разделителей текста `LangChain`.

25.2.6 Конвертировать тип

Компонент **Конвертировать тип** преобразует данные из одного типа в другой. Он поддерживает типы данных `Data`, `DataFrame` и `Message`.

Объект `Data` — это структурированный объект, содержащий основной ключ `text` и другие пары «ключ — значение»:

```
"data": {
  "text": "User Profile",
  "name": "Charlie Lastname",
  "age": 28,
  "email": "charlie.lastname@example.com"
}
```

Более широкий контекст, связанный со словарём data компонента, также определяет, какой ключ является основным (text_key), и может задать необязательное значение по умолчанию, если основной ключ не указан. Например:

```
{
  "text_key": "text",
  "data": {
    "text": "User Profile",
    "name": "Charlie Lastname",
    "age": 28,
    "email": "charlie.lastname@example.com"
  },
  "default_value": ""
}
```

DataFrame — это массив, представляющий табличную структуру данных со строками и столбцами.

Он состоит из списка (массива) словарных объектов, где каждый словарь представляет одну строку. Каждый ключ в словарях соответствует имени столбца. Например, следующий DataFrame содержит две строки со столбцами name, age и email:

```
[
  {
    "name": "Charlie Lastname",
    "age": 28,
    "email": "charlie.lastname@example.com"
  },
  {
    "name": "Bobby Othername",
    "age": 25,
    "email": "bobby.othername@example.com"
  }
]
```

Message в основном используется для передачи строки text, например "Name: Charlie Lastname, Age: 28, Email: charlie.lastname@example.com". Тем не менее полный объект Message может включать метаданные сообщения, особенно при использовании в качестве входных или выходных данных чата.

Для получения дополнительной информации см. Типы данных Генеративный ИИ.

Параметры компонента

Название	Отображаемое название	Описание
input_data	Входные данные	Входной параметр. Данные для преобразования. Принимает входные данные типов Data, DataFrame или Message.
output_type	Тип выхода	Входной параметр. Желаемый тип вывода: Данные, DataFrame или Сообщения.
output	Вывод	Выходной параметр. Преобразованные данные в указанном формате. Выходной порт меняется в

Название	Отображаемое название	Описание зависимости от выбранного Типа выхода.

25.3 Источник данных

25.3.1 API-запрос

Компонент **API-запрос** формирует и отправляет HTTP-запросы, используя URL-адреса или команды curl:

- **Режим URL:** Введите один или несколько URL-адресов, разделённых запятыми, а затем выберите метод запроса для каждого URL.
- **Режим curl:** Введите команду curl для выполнения.

Вы можете включить дополнительные параметры запроса в настройках компонента.

Возвращает объект Data, содержащий ответ.

Для API-компонентов, специфичных для конкретного провайдера, см. **Пакеты**.

Параметры API-запроса

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
mode	Режим	Входной параметр. Установите режим: URL или curl .
urls	URL	Входной параметр. Введите один или несколько URL-адресов, разделённых запятыми, для запроса.
curl	curl	Входной параметр. Только режим curl . Введите полную команду curl. Другие параметры компонента заполняются из аргументов команды.
method	Метод	Входной параметр. HTTP-метод, который следует использовать.
query_params	Параметры запроса	Входной параметр. Параметры запроса, которые следует добавить к URL.
body	Тело запроса	Входной параметр. Тело запроса, отправляемое с запросами POST, PATCH и PUT в виде словаря.

Название	Отображаемое название	Описание
headers	Заголовки	Входной параметр. Заголовки, отправляемые с запросом в виде словаря.
timeout	Тайм-аут	Входной параметр. Тайм-аут для выполнения запроса.
follow_redirects	Следовать перенаправлениям	Входной параметр. Следовать ли HTTP-перенаправлениям. Начиная с версии Генеративный ИИ 1.7, параметр Следовать перенаправлениям по умолчанию отключён (false) для предотвращения атак обхода SSRF, когда публичный URL перенаправляет на внутренние ресурсы. Включайте перенаправления только если доверяете целевому серверу. Подробнее см. переменные окружения защиты от SSRF.
save_to_file	Сохранить в файл	Входной параметр. Сохранять ли ответ API во временный файл. По умолчанию: false.
include_httpx_metadata	Включить метаданные HTTPx	Входной параметр. Включать ли в вывод такие свойства, как headers, status_code, response_headers и redirection_history. По умолчанию: false.

25.3.2 База знаний

База знаний Генеративный ИИ — это локальная векторная база данных, хранящаяся в хранилище Генеративный ИИ.

Поскольку базы знаний являются локальными, данные не запрашиваются удалённо и не переобрабатываются при каждом запуске потока. Это может быть эффективнее, чем использование удалённой векторной базы данных, и является хорошим выбором для потоков, работающих с пользовательскими предметно-ориентированными наборами данных — например, фрагментами данных о клиентах и продуктах.

Компоненты базы знаний можно использовать практически так же, как компоненты векторного хранилища. Однако есть несколько ключевых отличий:

- **Локальное хранение** — базы знаний Генеративный ИИ хранятся исключительно локально. Только некоторые компоненты векторного хранилища поддерживают локальные базы данных.
- **Встроенные модели эмбедингов** — базы знаний Генеративный ИИ включают встроенную поддержку нескольких моделей эмбедингов. Другие модели с базами знаний не поддерживаются. Чтобы использовать другого провайдера или модель, применяйте компонент векторного хранилища совместно с нужным компонентом модели эмбедингов.
- **Базовый поиск по сходству** — базы знаний Генеративный ИИ поддерживают только стандартный поиск по сходству. Для более сложных запросов используйте компонент векторного хранилища, подключённого к провайдеру с необходимой функциональностью.
- **Структурированные данные** — базы знаний Генеративный ИИ поддерживают только структурированные данные. Для неструктурированных данных используйте совместимый компонент векторного хранилища.

Компонент **База знаний** читает из существующей базы знаний с помощью семантического поиска.

Результатом вывода является объект DataFrame, содержащий наиболее релевантные результаты из запрошенной базы знаний.

Параметры компонента «База знаний»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
knowledge_base	База знаний	Входной параметр. Выберите базу знаний, из которой нужно получать данные.
api_key	Ключ API провайдера эмбедингов	Входной параметр. Необязательный ключ API для провайдера эмбедингов, позволяющий переопределить ранее указанный ключ. Провайдер и модель эмбедингов выбираются при создании базы знаний.
search_query	Поисковый запрос	Входной параметр. Необязательный запрос для фильтрации данных базы знаний по семантическому сходству. Если не указан, возвращаются наиболее

Название	Отображаемое название	Описание
top_k	Количество результатов	релевантные результаты в произвольном порядке. Входной параметр. Количество возвращаемых результатов поиска. По умолчанию: 5.
include_metadata	Включать метаданные	Входной параметр. Определяет, включать ли все метаданные и эмбединги в вывод. Если включено, каждая строка вывода содержит все метаданные, эмбединги и контент. Если отключено, возвращается только контент. По умолчанию: true.

Использование компонента «База знаний» в потоке

После создания базы знаний и загрузки в неё данных (см. Управление векторными данными) вы можете использовать компонент **База знаний** в любом потоке для семантического поиска по её содержимому:

1. Добавьте компонент **База знаний** в ваш поток.
2. В поле **База знаний** выберите нужную базу — например, базу данных о продажах клиентов, созданную на предыдущих шагах.
3. Чтобы просмотреть результаты поиска как сообщения чата, подключите вывод **Результаты** к компоненту **Вывод чата**.
4. В поле **Поисковый запрос** введите запрос, соответствующий данным в базе знаний. Для примера с данными о продажах клиентов введите название продукта, например laptop или wireless devices.
5. Нажмите **Запустить компонент** на компоненте **База знаний**, а затем откройте **Playground** для просмотра результатов.

25.3.3 Чтение файла

В версии Генеративный ИИ 1.7.0 этот компонент был переименован с **Файл** на **Чтение файла**.

Компонент **Чтение файла** загружает и разбирает файлы, преобразуя их содержимое в объект Data, DataFrame или Message. Он поддерживает множество типов файлов, предоставляет параметры для параллельной обработки и обработки ошибок, а также расширенный разбор с помощью библиотеки Docling.

Файлы можно добавить в компонент **Чтение файла** в визуальном редакторе или во время выполнения потока, а также загрузить несколько файлов одновременно. Для получения дополнительной информации о загрузке файлов и работе с ними в потоках см. Управление файлами и Создание чат-бота, способного обрабатывать файлы.

Компонент **Чтение файла** может читать файлы из локального хранилища базы данных Генеративный ИИ, AWS S3 или **Google Drive**. Для получения дополнительной информации см. Настройка хранилища файлов.

Типы файлов и ограничения по размеру

По умолчанию максимальный размер файла составляет 1024 МБ. Чтобы изменить это значение, задайте переменную окружения `ASTRA_MULTIAGENT_AI_MAX_FILE_SIZE_UPLOAD` в разделе Переменные окружения.

Поддерживаемые типы файлов

Следующие типы файлов поддерживаются компонентом **Чтение файла**. Используйте архивные и сжатые форматы для объединения нескольких файлов или используйте компонент **Папка** для загрузки всех файлов из каталога.

- .bz2
- .csv
- .docx
- .gz
- .htm
- .html
- .json
- .js
- .md
- .mdx
- .pdf
- .py
- .sh
- .sql
- .tar
- .tgz
- .ts
- .tsx
- .txt
- .xml
- .yaml
- .yml
- .zip

Если необходимо загрузить файл неподдерживаемого типа, используйте другой компонент, поддерживающий данный тип, и при необходимости выполните разбор за пределами Генеративный ИИ, либо преобразуйте файл в поддерживаемый тип перед загрузкой.

Для изображений см. Загрузка изображений.

Для видео см. пакеты **Twelve Labs** и **YouTube**.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
path	Файлы	Входной параметр. Путь к файлам для загрузки.

Название	Отображаемое название	Описание
		Может быть локальным или в хранилище файлов Генеративный ИИ. Поддерживает отдельные файлы и архивы.
file_path	Серверный путь к файлу	Входной параметр. Объект Data со свойством file_path, указывающим на файл в хранилище файлов Генеративный ИИ, или объект Message с путём к файлу. Заменяет параметр Файлы (path), но поддерживает те же типы файлов.
separator	Разделитель	Входной параметр. Разделитель, используемый между несколькими выводами в формате Message.
silent_errors	Тихие ошибки	Входной параметр. Если true, ошибки компонента не вызывают исключение. По умолчанию: false.
delete_server_file_after_processing	Удалить серверный файл после обработки	Входной параметр. Если true (по умолчанию), файл по пути Серверный путь к файлу (file_path) удаляется после обработки.
ignore_unsupported_extensions	Игнорировать неподдерживаемые расширения	Входной параметр. Если true, файлы с неподдерживаемыми расширениями принимаются, но не обрабатываются. Если false, компонент Чтение файла выдаёт ошибку при получении файла неподдерживаемого типа. По умолчанию: true.
ignore_unspecified_files	Игнорировать неуказанные файлы	Входной параметр. Если true, объекты Data без свойства file_path

Название	Отображаемое название	Описание
concurrency_multithreading	Параллельная обработка	игнорируются. Если false (по умолчанию), компонент выдаёт ошибку, если файл не указан. Входной параметр. Количество файлов, обрабатываемых одновременно при загрузке нескольких файлов. По умолчанию: 1. Значения больше 1 включают параллельную обработку для двух и более файлов. Игнорируется при загрузке одного файла и при расширенном разборе.
advanced_parser	Расширенный парсер	Входной параметр. Если true, включает расширенный разбор. Доступно только для одного файла совместимого типа. По умолчанию: false.

Расширенный разбор

Начиная с версии Генеративный ИИ 1.6, компонент **Чтение файла** поддерживает расширенный разбор документов с использованием библиотеки Docling для поддерживаемых типов файлов.

Чтобы использовать расширенный разбор:

1. Выполните следующие предварительные требования, если применимо:

- **Установите Генеративный ИИ версии 1.6 или выше:** более ранние версии не поддерживают расширенный разбор с компонентом **Чтение файла**. Руководство по обновлению см. в Примечаниях к выпуску.
- **Установите зависимость Docling на macOS Intel (x86_64):** зависимость Docling не устанавливается по умолчанию для macOS Intel (x86_64). Используйте руководство по установке Docling для установки зависимости. Для всех остальных операционных систем зависимость Docling устанавливается по умолчанию.
- **Системные зависимости Docker/Linux:** если вы запускаете Генеративный ИИ в контейнере Docker на Linux, возможно, потребуется установить дополнительные системные пакеты для обработки документов. Подробнее см. в разделе Ошибки обработки документов в Docker-контейнерах.
- **Включите режим разработчика на Windows:** Если вы используете Генеративный ИИ Desktop на Windows, необходимо включить режим разработчика для использования компонентов Docling. Расположение этой настройки зависит от версии Windows. Найдите раздел **Для разработчиков** в настройках Windows или выполните поиск по слову

«Developer» в строке поиска, затем включите **Режим разработчика**. Возможно, потребуется перезагрузить компьютер или Генеративный ИИ, чтобы изменения вступили в силу.

Режим разработчика не требуется для модуля «Генеративный ИИ» OSS на Windows.

2. Добавьте один файл совместимого типа в компонент **Чтение файла**.

- Расширенный разбор обрабатывает только один файл. Если выбрано несколько файлов, компонент **Чтение файла** обработает только первый, игнорируя остальные. Чтобы обработать несколько файлов с расширенным разбором, передайте каждый файл в отдельный компонент **Чтение файла** или используйте специализированные компоненты **Docling**.
- Расширенный разбор поддерживает все типы файлов компонента **Чтение файла**, за исключением .csv, .xlsx и .parquet, поскольку он предназначен для обработки документов, например извлечения текста из PDF. Для анализа структурированных данных используйте компонент **Парсер**.

3. Включите **Расширенный парсер**.

3. Чтобы настроить параметры расширенного разбора, выберите компонент, чтобы открыть панель инспекции компонента.

Название	Отображаемое название	Описание
pipeline	Пайплайн	Входной параметр, расширенный разбор. Конвейер Docling: standard (по умолчанию, рекомендуется) или vlm (может давать нестабильные результаты).
ocr_engine	Механизм OCR	Входной параметр, расширенный разбор. OCR-движок для конвейера standard. Варианты: None (по умолчанию) или EasyOCR. Значение None означает, что OCR не используется, что может приводить к нестабильным или ошибочным результатам для некоторых документов. Не влияет на конвейер vlm.
md_image_placeholder	Заполнитель изображения Markdown	Входной параметр, расширенный разбор. Заполнитель для изображений при выводе в формате Markdown . По умолчанию: <!-- image -->.

Название	Отображаемое название	Описание
md_page_break_placeholder	Заполнитель разрыва страницы Markdown	Входной параметр, расширенный разбор. Заполнитель для разрывов страниц при выводе в формате Markdown . По умолчанию: "" (пустая строка).
doc_key	Ключ документа	Входной параметр, расширенный разбор. Ключ для столбца DoclingDocument, содержащего структурированную информацию, извлечённую из исходного документа. Подробнее см. DoclingDocument. По умолчанию: doc.

Для дополнительных функций Docling, включая другие компоненты и OCR-движки, используйте пакет **Docling**.

Вывод компонента

Вывод компонента **Чтение файла** зависит от количества загруженных файлов и от того, включён ли расширенный разбор. Если доступно несколько вариантов, тип вывода можно выбрать рядом с выходным портом компонента.

Если запустить компонент **Чтение файла** без выбранного файла, он выдаст ошибку или, если включены **Тихие ошибки**, не сгенерирует вывод.

Если расширенный разбор отключён и загружен один файл, доступны следующие типы вывода:

- **Структурированное содержимое:** доступно только для файлов .csv, .xlsx, .parquet и .json.
 - Для файлов .csv генерирует DataFrame, представляющий табличные данные.
 - Для файлов .json генерирует объект Data с разобранными данными JSON.
- **Сырой контент:** объект Message, содержащий необработанное текстовое содержимое файла.
- **Путь к файлу:** объект Message, содержащий путь к файлу в хранилище файлов Генеративный ИИ.

Если расширенный разбор включён и загружен один файл, доступны следующие типы вывода:

- **Структурированный вывод:** объект DataFrame, содержащий данные документа, обработанные Docling, с текстовыми элементами, номерами страниц и метаданными.
- **Markdown:** объект Message, содержащий содержимое загруженного документа в формате Markdown с заполнителями изображений.
- **Путь к файлу:** объект Message, содержащий путь к файлу в хранилище файлов Генеративный ИИ.

Если загружено несколько файлов, компонент выводит **Файлы** — DataFrame, содержащий содержимое и метаданные всех выбранных файлов.

Расширенный разбор не поддерживает несколько файлов и обрабатывает только первый файл.

25.3.4 Записать в файл

В версии Генеративный ИИ 1.7.0 этот компонент был переименован из **Сохранить файл** в **Записать в файл**.

Компонент **Записать в файл** создаёт файл с данными, полученными от другого компонента. Поддерживается несколько форматов файлов, а сохранять файлы можно в хранилище файлов Генеративный ИИ, AWS S3, Google Drive или локальную файловую систему.

Чтобы настроить компонент **Записать в файл** и использовать его в потоке, выполните следующие действия:

1. Подключите выход DataFrame, Data или Message другого компонента к входному порту **Ввод компонента Записать в файл**.
Один и тот же выход можно подключить к нескольким компонентам **Записать в файл**, если вы хотите создать несколько файлов, сохранить данные в разных форматах или записать их в разные места.
2. Нажмите **Место хранения** и выберите **Local**, **AWS** или **Google Drive**.
При необходимости введите учётные данные облачного провайдера. Для получения дополнительной информации см. Настройка хранилища файлов.
3. В поле **Имя файла** укажите имя файла и, при необходимости, путь к нему.
Параметр **Имя файла** определяет, где будет сохранён файл. Он может содержать только имя файла или полный путь к нему:
 - **Расположение по умолчанию**: если указать только имя файла, файл будет сохранён в каталоге данных Генеративный ИИ. Например, ~/Library/Caches/Генеративный ИИ/data на macOS.
 - **Подкаталог**: чтобы сохранять файлы в подкаталоге, добавьте путь в параметр **Имя файла**. Если указанный подкаталог ещё не существует, Генеративный ИИ создаст его автоматически. Например, files/my_file создаст файл my_file в /data/files и при необходимости создаст каталог files.
 - **Абсолютный или относительный путь**: чтобы сохранить файл в другом месте вашей среды или локального хранилища, укажите абсолютный или относительный путь. Например, ~/Desktop/my_file сохранит файл my_file на рабочем столе.

Не указывайте расширение в имени файла. Если вы его добавите, оно будет считаться частью имени и не повлияет на параметр **Формат файла**.

4. Выберите нужный **Формат файла** и нажмите **Заккрыть**.
Доступные форматы зависят от типа входных данных:
 - DataFrame можно сохранить в CSV (по умолчанию), Excel (требуется openpyxl как дополнительная зависимость), JSON (резервный формат по умолчанию) или Markdown.
 - Data можно сохранить в CSV, Excel (требуется openpyxl как дополнительная зависимость), JSON (по умолчанию) или Markdown.
 - Message можно сохранить в TXT, JSON (по умолчанию) или Markdown.

Если в одном или нескольких потоках используется несколько компонентов **Записать в файл** с одинаковыми именем файла, путём и расширением, файл будет содержать данные только от последнего запуска. Генеративный ИИ не блокирует перезапись, если файл с таким именем уже существует. Чтобы избежать случайной перезаписи, используйте уникальные имена файлов и пути.

5. Чтобы протестировать компонент **Записать в файл**, нажмите **Run component**, а затем **Inspect output**, чтобы получить путь к сохранённому файлу.

Фактический вывод компонента — это объект Message, содержащий исходный тип данных, имя файла и расширение, а также абсолютный путь к файлу, сформированный на основе параметра **Имя файла**.

Например:

```
DataFrame saved successfully as 'my_file.csv' at /Users/user.name/Library/Caches/Генеративный ИИ/data/my_file.csv
```

Если параметр **Имя файла** содержит подкаталог или другой путь, отличный от пути по умолчанию, это также будет отражено в выводе Message.

Например, для CSV-файла с именем ~/Desktop/my_file вывод может выглядеть так:

```
DataFrame saved successfully as '/Users/user.name/Desktop/my_file.csv' at /Users/user.name/Desktop/my_file.csv
```

5. Необязательно: если вы хотите использовать сохранённый файл в потоке, выполните API-вызов или используйте другой компонент, чтобы получить файл по указанному пути.

25.4 Файлы и данные

25.4.1 Папка

Компонент **Папка** рекурсивно загружает файлы из каталога, с параметрами для типов файлов, глубины и параллелизма.

Файлы должны быть поддерживаемого типа и размера для загрузки.

Возвращает либо объект Data, либо объект DataFrame в зависимости от содержимого каталога.

Папка параметры

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
path	MessageTextInput	Входной параметр. Путь к каталогу, из которого загружаются файлы. По умолчанию: текущий каталог (.)
types	MessageTextInput	Входной параметр. Типы файлов для загрузки. Выберите один или несколько, либо оставьте пустым, чтобы попытаться загрузить все файлы.

Название	Тип	Описание
depth	IntInput	Входной параметр. Глубина поиска файлов.
max_concurrency	IntInput	Входной параметр. Максимальная параллельность при загрузке нескольких файлов.
load_hidden	BoolInput	Входной параметр. Если true, скрытые файлы загружаются.
recursive	BoolInput	Входной параметр. Если true, поиск будет рекурсивным.
silent_errors	BoolInput	Входной параметр. Если true, ошибки не вызывают исключение.
use_multithreading	BoolInput	Входной параметр. Если true, используется многопоточность.

25.4.2 Тестовые данные

Компонент **Тестовые данные** генерирует примерные данные для тестирования и разработки. Доступны следующие типы вывода:

- `message_output` — вывод сообщения (текст) с примерным текстом Lorem Ipsum.
- `data_output` — объект данных (JSON), содержащий JSON-структуру с одной примерной записью в `records` и секцией `summary`.
- `dataframe_output` — `DataFrame` (табличный) с 50 тестовыми записями, включая столбцы `customer_id`, `first_name` и `last_name`.

25.4.3 URL

Компонент **URL** получает содержимое по одному или нескольким URL-адресам, обрабатывает его и возвращает в разных форматах.

Он рекурсивно переходит по ссылкам до заданной глубины и поддерживает вывод в виде обычного текста, Markdown или необработанного HTML.

Параметры URL

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые из доступных параметров:

Название	Отображаемое название	Описание
urls	URL-адреса	Входной параметр. Один или несколько URL-адресов для рекурсивного обхода. В визуальном редакторе нажмите

Название	Отображаемое название	Описание
max_depth	Глубина	Добавить URL, чтобы указать несколько адресов. Входной параметр. Управляет глубиной обхода ссылок, то есть тем, сколько «кликов» от начальной страницы сделает сканер. Глубина 1 ограничивает обход только первой страницей по указанному URL. Глубина 2 означает, что сканер обходит первую страницу и каждую страницу, напрямую связанную с ней, после чего останавливается. Эта настройка управляет только обходом ссылок и не ограничивает количество сегментов в URL или домен.
prevent_outside	Не выходить за пределы домена	Входной параметр. Если включено, компонент обходит только URL-адреса в том же домене, что и исходный URL. Это предотвращает переход на внешние сайты, даже если на них ведут ссылки с уже обработанных страниц.
use_async	Использовать асинхронный режим	Входной параметр. Если включено, используется асинхронная загрузка. Она может работать значительно быстрее, но может потреблять больше системных ресурсов.
format	Формат вывода	Входной параметр. Определяет формат вывода: Текст, Markdown или HTML. По умолчанию — Текст. Для получения дополнительной

Название	Отображаемое название	Описание
		информации см. Вывод URL.
timeout	Тайм-аут	Входной параметр. Тайм-аут запроса в секундах.
headers	Заголовки	Входной параметр. Заголовки, отправляемые вместе с запросом, если они требуются для аутентификации или по другим причинам.

Дополнительные входные параметры доступны для обработки ошибок и управления кодировкой.

Вывод URL

Существуют два параметра, которые управляют выводом компонента **URL** на разных этапах:

- **Формат вывода:** этот необязательный параметр управляет содержимым, извлекаемым из обработанных страниц:
- **Текст** (по умолчанию): компонент извлекает только текст из HTML обработанных страниц.
- **Markdown:** компонент преобразует HTML-контент в формат Markdown с помощью Markitdown.
- **HTML:** компонент извлекает весь необработанный HTML-контент обработанных страниц.
- **Тип выходных данных:** в поле вывода компонента, рядом с выходным портом, можно выбрать структуру исходящих данных при передаче в другие компоненты:
- **Extracted Pages:** возвращает DataFrame, в котором обработанные страницы разбиты на столбцы с полным содержимым страницы (text) и метаданными, такими как url и title.
- **Raw Content:** возвращает Message, содержащий весь текст, Markdown или HTML из обработанных страниц вместе с метаданными в виде одного текстового блока.

При использовании в обычном потоке компонент **URL** должен быть подключён к компоненту, который принимает выбранный тип выходных данных, то есть DataFrame или Message.

Вы можете подключить компонент **URL** напрямую к совместимому компоненту или использовать компонент **Конвертировать тип**, чтобы преобразовать вывод в другой тип перед передачей данных, если типы данных несовместимы.

Обрабатывающие компоненты, такие как **Конвертировать тип**, особенно полезны вместе с компонентом **URL**, поскольку он может извлекать большие объёмы данных из обработанных страниц.

Например, если вам нужно передать дальше только отдельные поля, вы можете использовать компонент **Парсер**, чтобы извлечь только эти данные из обработанных страниц перед их передачей другим компонентам.

При использовании в **Режиме инструмента** с компонентом **Агент** компонент **URL** можно подключить напрямую к порту **Инструменты** компонента **Агент** без преобразования данных.

Агент сам решает, нужно ли использовать компонент **URL** в зависимости от запроса пользователя, и может напрямую обработать вывод типа **DataFrame** или **Message**.

25.4.4 Веб-поиск

Компонент **Веб-поиск** объединяет компоненты **Веб-поиск**, **Поиск новостей** и **RSS-ридер** в один компонент с вкладками для разных режимов поиска. С его помощью можно выполнять веб-поиск через **DuckDuckGo**, искать новости в **Google News** и читать **RSS-ленты** из одного компонента.

Для других поисковых API см. **Пакеты**.

Компонент **Веб-поиск** использует веб-скрейпинг, поэтому его работа может ограничиваться лимитами запросов.

Для использования в **production**-среде рекомендуется выбрать другой поисковый компонент с более надёжной API-поддержкой, например специализированные пакеты конкретных поставщиков.

Использование в потоке

Следующие шаги показывают один из способов использования компонента **Веб-поиск** в потоке:

1. Создайте поток на основе шаблона **Basic Prompting**.
2. Добавьте компонент **Веб-поиск**, выберите нужный **Режим поиска** (**Web**, **News** или **RSS**), а затем введите поисковый запрос или **URL RSS-ленты**.
3. Добавьте компонент **Конвертировать тип**, установите **Тип выхода** в **Сообщения**, а затем подключите выход компонента **Веб-поиск** ко входу компонента **Конвертировать тип**.

По умолчанию компонент **Веб-поиск** возвращает **DataFrame**. Поскольку компонент **Шаблон промпта** принимает только данные типа **Message**, это преобразование необходимо, чтобы передать результаты поиска в компонент **Шаблон промпта**. Для получения дополнительной информации см. **Вывод компонента «Веб-поиск»**.

4. В поле **Шаблон** компонента **Шаблон промпта** добавьте переменную, например **{searchresults}** или **{context}**.

Это добавит в компонент **Шаблон промпта** поле переменной, через которое можно передать преобразованные результаты поиска в запрос. Для получения дополнительной информации см. **Определение переменных в промптах**.

5. Подключите выход компонента **Конвертировать тип** к новому полю переменной в компоненте **Шаблон промпта**.
6. В компоненте **Языковая модель** добавьте API-ключ **OpenAI** или выберите другого поставщика и модель.
7. Нажмите **Playground**, а затем введите запрос.

LLM обработает запрос с учётом контекста, переданного через компонент **Шаблон промпта**, и выведет ответ в чат-интерфейсе **Playground**.

Параметры

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
search_mode	Режим поиска	Входной параметр. Позволяет выбрать режим

Название	Отображаемое название	Описание
		поиска: Web (DuckDuckGo), News (Google News) или RSS (Feed Reader). По умолчанию: Web.
query	Поисковый запрос	Входной параметр. Ключевые слова для поиска.
timeout	Тайм-аут	Входной параметр. Тайм-аут веб-запроса в секундах. По умолчанию: 5.
results	Результаты	Выходной параметр. Возвращает DataFrame, содержащий столбцы title, link, snippet и content. Подробнее см. Вывод компонента «Веб-поиск».

Название	Отображаемое название	Описание
search_mode	Режим поиска	Входной параметр. Позволяет выбрать режим поиска: Web (DuckDuckGo), News (Google News) или RSS (Feed Reader). По умолчанию: Web.
query	Поисковый запрос	Входной параметр. Ключевые слова для поиска новостных статей.
hl	Язык	Входной параметр. Код языка, например en-US, fr, de. По умолчанию: en-US.
gl	Страна	Входной параметр. Код страны, например US, FR, DE. По умолчанию: US.
ceid	Страна: язык	Входной параметр. Пара язык-страна, например US:en, FR:fr. По умолчанию: US:en.
topic	Тема	Входной параметр. Одно из значений: WORLD, NATION, BUSINESS, TECHNOLOGY, ENTERTAINMENT, SCIENCE, SPORTS, HEALTH.

Название	Отображаемое название	Описание
location	Местоположение	Входной параметр. Город, штат или страна для новостей по географическому признаку. Оставьте пустым для поиска по ключевым словам.
timeout	Тайм-аут	Входной параметр. Тайм-аут запроса в секундах. По умолчанию: 5.
results	Результаты	Выходной параметр. Возвращает DataFrame с ключевыми столбцами title, link, published и summary. Подробнее см. Вывод компонента «Веб-поиск».

Название	Отображаемое название	Описание
search_mode	Режим поиска	Входной параметр. Позволяет выбрать режим поиска: Web (DuckDuckGo), News (Google News) или RSS (Feed Reader). По умолчанию: Web.
query	URL RSS-ленты	Входной параметр. URL RSS-ленты для разбора, например https://rss.nytimes.com/services/xml/rss/nyt/HomePage.xml .
timeout	Тайм-аут	Входной параметр. Тайм-аут запроса RSS-ленты в секундах. По умолчанию: 5.
results	Результаты	Выходной параметр. Возвращает DataFrame, содержащий ключевые столбцы title, link, published и summary. Подробнее см. Вывод компонента «Веб-поиск».

Вывод компонента «Веб-поиск»

Компонент **Веб-поиск** возвращает DataFrame, состав столбцов которого зависит от выбранного режима поиска.

При использовании режима **Web** компонент возвращает DataFrame со следующими столбцами:

- title: заголовок результата поиска
- link: URL результата поиска
- snippet: краткий фрагмент результата поиска
- content: полное содержимое страницы, если его удалось получить

При использовании режима **News** компонент возвращает DataFrame со следующими столбцами:

- title: заголовок новостной статьи
- link: URL новостной статьи
- published: дата публикации статьи
- summary: краткое описание или содержание статьи

При использовании режима **RSS** компонент возвращает DataFrame со следующими столбцами:

- title: заголовок элемента RSS-ленты
- link: URL элемента RSS-ленты
- published: дата публикации элемента
- summary: краткое описание или содержание элемента

25.5 Управление потоком

25.5.1 Если-иначе

Компонент **Если-иначе** — это условный маршрутизатор, который направляет сообщения, сравнивая две строки. Он оценивает условие путём сравнения двух входных текстов с помощью указанного оператора, а затем направляет сообщение в `true_result` или `false_result` в зависимости от результата.

Оператор ищет отдельные строки во входных данных (`input_text`) на основе заданного оператора и сопоставляемого текста (`match_text`), но также поддерживает поиск по нескольким словам с помощью регулярного выражения.

Доступные операторы:

- **equals** — точное совпадение строк
- **not equals** — обратное условие точного совпадения
- **contains** — проверяет, содержится ли `match_text` внутри `input_text`
- **starts with** — проверяет, начинается ли `input_text` с `match_text`
- **ends with** — проверяет, заканчивается ли `input_text` на `match_text`
- **regex** — сопоставление по регистрозависимому шаблону

По умолчанию все операторы регистронезависимы, за исключением **regex**. **regex** всегда регистрозависим. Включить регистрозависимость для остальных операторов можно в разделе Параметры компонента.

Использование компонента Если-иначе в потоке

Следующий пример демонстрирует использование компонента **Если-иначе** для проверки входящих сообщений чата с помощью регулярного выражения и вывода разного ответа в зависимости от результата сопоставления.

1. Добавьте компонент **Если-иначе** в поток и настройте его:

- **Вход текста** — подключите порт **Вход текста** к компоненту **Ввод чата** или другому источнику с типом `Message`.

Если ваш ввод не в формате Message, используйте компонент **Конвертировать тип** или **Парсер** для преобразования. Если ваш ввод в принципе не подходит для формата Message, рассмотрите компонент **Операции с данными** для условной маршрутизации.

- **Текст совпадения** — введите `.*(urgent|warning|caution).*`, чтобы компонент искал эти значения во входных данных. Сопоставление по regex регистрозависимо, поэтому для поиска всех вариантов написания слова warning введите `warning|Warning|WARNING`.
- **Оператор** — выберите **regex**.
- **Случай верен** — в панели инспекции компонента включите параметр **Случай верен**, нажмите **Заккрыть**, затем введите `New Message Detected` в соответствующем поле.

Сообщение **Случай верен** отправляется через выход **True**, когда условие сопоставления истинно. Сообщение для **Случай неверен** не задано — компонент не генерирует вывод, когда условие ложно.

2. Добавьте компоненты для обработки результата **True**:

1. Добавьте в поток компоненты **Языковая модель**, **Шаблон промпта** и **Вывод чата**.
2. В компоненте **Языковая модель** введите ключ API OpenAI или выберите другого провайдера и модель.
3. Подключите выход **True** компонента **Если-иначе** к порту **Вход** компонента **Языковая модель**.
4. В компоненте **Шаблон промпта** введите инструкции для модели при истинном результате.
5. Подключите компонент **Шаблон промпта** к порту **Системное сообщение** компонента **Языковая модель**.
6. Подключите выход компонента **Языковая модель** к компоненту **Вывод чата**.

3. Повторите тот же процесс с отдельным набором компонентов **Языковая модель**, **Шаблон промпта** и **Вывод чата** для результата **False**.

Подключите выход **False** компонента **Если-иначе** к порту **Вход** второго компонента **Языковая модель**. Во втором **Шаблон промпта** введите инструкции для ложного результата, например: `Send a message that a new low-priority message was received`.

4. Чтобы протестировать поток, откройте **Playground** и отправьте несколько сообщений — с вхождением строк из regex и без них. Вывод чата должен отражать инструкции из промпта в зависимости от результата проверки:

Параметры компонента

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Изменить все параметры можно через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
input_text	Строка	Входной параметр. Основной текст для операции сравнения.
match_text	Строка	Входной параметр. Текст для сравнения.

Название	Тип	Описание
operator	Выпадающий список	Входной параметр. Оператор сравнения текстов. Варианты: equals, not equals, contains, starts with, ends with, regex. По умолчанию: equals.
case_sensitive	Булево	Входной параметр. Если true, сравнение учитывает регистр. По умолчанию: false. Не применяется к оператору regex.
max_iterations	Целое	Входной параметр. Максимальное количество итераций условного маршрутизатора. По умолчанию: 10.
default_route	Выпадающий список	Входной параметр. Маршрут при достижении максимального числа итераций. Варианты: true_result, false_result. По умолчанию: false_result.
true_result	Сообщение	Выходной параметр. Вывод, когда условие истинно.
false_result	Сообщение	Выходной параметр. Вывод, когда условие ложно.

25.5.2 Цикл

Компонент **Цикл** перебирает список входных данных, передавая отдельные элементы компонентам, подключённым к выходному порту **Элемент**, пока не останется элементов для обработки. После этого компонент **Цикл** передаёт агрегированный результат всех итераций компоненту, подключённому к порту **Готово**.

Процесс итерации

Компонент **Цикл** работает как миниатюрный вложенный поток. Ниже приведено описание процесса итерации:

1. Компонент принимает список объектов Data или DataFrame — например, CSV-файл — через входной порт **Вход**.
2. Входные данные разбиваются на отдельные элементы. Например, CSV-файл разбивается построчно.

Компонент **Цикл** последовательно извлекает элементы по ключу text из объектов Data или DataFrame, пока все элементы не будут обработаны. Каждый выводимый элемент item представляет собой объект Data.

3. Компонент итерирует по каждому элементу `item`, передавая его в выходной порт **Элемент**.

Этот порт подключается к одному или нескольким компонентам, выполняющим обработку каждого элемента. Последний компонент в цепочке подключается обратно к порту **Цикл** компонента **Цикл**, чтобы инициировать обработку следующего элемента.

К порту **Элемент** подключается только один компонент, однако данные можно передавать через любое количество компонентов — при условии, что последний из них подключается обратно к порту **Цикл**.

Компонент **Если-иначе** несовместим с компонентом **Цикл**. Для получения дополнительной информации см. Условный цикл.

4. После обработки всех элементов результаты агрегируются в единый объект `Data`, который передаётся из порта **Готово** компонента **Цикл** в следующий компонент потока.

Следующий упрощённый код на Python иллюстрирует работу компонента **Цикл**. Это *не* реальный код компонента — он предназначен только для понимания общего процесса.

```
for i in input:           # Receive input data as a list
    process_item(i)       # Process each item through components connected at the Item port
    if has_more_items():
        continue         # Loop back to Looping port to process the next item
    else:
        break            # Exit the loop when no more items are left

done = aggregate_results() # Compile all returned items

print(done)              # Send the aggregated results from the Done port to another component
```

Пример использования компонента «Цикл»

В следующем примере компонент **Цикл** перебирает строки CSV-файла до тех пор, пока не останется необработанных строк. Порт **Элемент** передаёт каждую строку компоненту **Конвертировать тип**, который преобразует её в объект `Message`. Затем `Message` передаётся компоненту **Структурированный вывод** для формирования структурированных данных, которые возвращаются к порту **Цикл** компонента **Цикл**. После обработки всех строк компонент **Цикл** загружает агрегированный список структурированных данных в базу данных `Chroma DB` через компонент **Chroma DB**, подключённый к порту **Готово**.

Для получения дополнительных примеров использования компонента **Цикл** попробуйте шаблон **Research Translation Loop** в модуле «Генеративный ИИ» или посмотрите видеоруководство Освоение компонента «Цикл» и `Agentic RAG` в модуле «Генеративный ИИ».

Условный цикл

Компонент **Если-иначе** несовместим с компонентом **Цикл**. Если требуется условная логика внутри цикла, перестройте поток так, чтобы условия обрабатывались до начала итерации. Например, при переборе `DataFrame` можно использовать несколько компонентов **Операции с DataFrame** для предварительной условной фильтрации данных, а затем запускать отдельные циклы для каждого набора отфильтрованных данных.

25.5.3 Уведомления и прослушивание

Компоненты **Уведомление** и **Прослушивание** используются в паре.

Компонент **Уведомление** формирует уведомление из контекста текущего потока, включая конкретное содержимое данных и идентификатор статуса.

Сформированное уведомление передаётся в компонент **Прослушивание**.

Данные уведомления затем можно передать другим компонентам в потоке, например компоненту **Если-иначе**.

25.5.4 Запустить поток

Компонент **Запустить поток** запускает другой поток Генеративный ИИ как подпроцесс текущего потока.

Этот компонент можно использовать для последовательного соединения потоков, условного запуска потоков и подключения потоков к компонентам **Агент** в качестве инструментов, вызываемых по необходимости.

При использовании с агентом метаданные name и description, которые агент использует для регистрации инструмента, создаются автоматически.

Когда вы выбираете поток для компонента **Запустить поток**, он использует структуру графа целевого потока для динамического создания полей ввода и вывода.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
flow_name_selected	Выпадающий список	Входной параметр. Имя потока для запуска.
session_id	String	Входной параметр. Идентификатор сеанса для выполнения потока, если вы хотите передать пользовательский идентификатор сеанса в подпоток.
flow_tweak_data	Dict	Входной параметр. Словарь настроек для изменения поведения потока. Доступные настройки зависят от выбранного потока.
dynamic inputs	Различные	Входной параметр. Дополнительные поля ввода, генерируемые на основе структуры выбранного потока.
run_outputs	List[Data \ Message \ DataFrame]	Выходной параметр. Все выходные данные, возвращаемые при выполнении потока.

25.6 LLM операторы

25.6.1 Пакетный запуск

Компонент **Пакетный запуск** запускает языковую модель для *каждой строки одного текстового столбца* в DataFrame и возвращает новый DataFrame с исходным текстом и ответом LLM. Вывод содержит следующие столбцы:

- text_input: Исходный текст из входного DataFrame
- model_response: Ответ модели для каждого входа
- batch_index: Порядковый номер обработки с нулевой индексацией для всех строк в DataFrame
- metadata (опционально): Дополнительная информация о процессе обработки

Использование компонента «Пакетный запуск» в потоке

Если передать вывод компонента **Пакетный запуск** в компонент **Парсер**, вы можете использовать переменные в шаблоне парсинга для обращения к этим ключам, например {text_input} и {model_response}. Это продемонстрировано в следующем примере.

1. Подключите любой компонент языковой модели к порту **Языковая модель** компонента **Пакетный запуск**.
2. Подключите вывод DataFrame из другого компонента к входу **DataFrame** компонента **Пакетный запуск**. Например, вы можете подключить компонент **Чтение файла с CSV-файлом**.
3. В поле **Имя столбца** компонента **Пакетный запуск** введите название столбца во входном DataFrame, содержащего текст для обработки. Например, если вы хотите извлечь текст из столбца name в CSV-файле, введите name в поле **Имя столбца**.
4. Подключите вывод **Batch Results** компонента **Пакетный запуск** к входу **DataFrame** компонента **Парсер**.
5. Опционально: в меню компонента **Пакетный запуск** включите параметр **Системное сообщение**, нажмите **Заккрыть**, а затем введите инструкцию о том, как LLM должна обрабатывать каждую ячейку, извлечённую из файла. Например: Create a business card for each name.
6. В поле **Шаблон** компонента **Парсер** введите шаблон для обработки новых столбцов DataFrame компонента **Пакетный запуск** (text_input, model_response и batch_index):
Например, этот шаблон использует три столбца из итогового DataFrame после пакетной обработки:
record_number: {batch_index}, name: {text_input}, summary: {model_response}
7. Чтобы протестировать обработку, нажмите на компонент **Парсер**, затем — **Запустить компонент**, а затем — **Проверка вывода**, чтобы просмотреть итоговый DataFrame.
Вы также можете подключить компонент **Вывод чата** к компоненту **Парсер**, если хотите увидеть вывод в **Playground**.

Параметры пакетного запуска

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
model	HandleInput	Входной параметр. Подключите вывод «Языковая модель» из компонента языковой модели. Обязательно.

Название	Тип	Описание
system_message	MultilineInput	Входной параметр. Многострочная системная инструкция для всех строк в DataFrame.
df	DataFrameInput	Входной параметр. DataFrame, столбец которого рассматривается как текстовые сообщения, указанные в column_name. Обязательно.
column_name	MessageTextInput	Входной параметр. Имя столбца DataFrame, который следует рассматривать как текстовые сообщения. Если пусто, все столбцы форматируются в TOML.
output_column_name	MessageTextInput	Входной параметр. Название столбца, где хранится ответ модели. По умолчанию: model_response.
enable_metadata	BoolInput	Входной параметр. Если true, добавить метаданные в выходной DataFrame.
batch_results	DataFrame	Выходной параметр. DataFrame со всеми исходными столбцами плюс столбец с ответом модели.

25.6.2 Ограничения

Компонент **Ограничения** проверяет входной текст на соответствие правилам безопасности и защиты, отправляя запросы в языковую модель (LLM) для проверки нарушений.

Можно проверять следующие ограничения:

- **PII** — обнаруживает персональные идентифицируемые данные: имена, адреса, номера телефонов, адреса электронной почты, номера социального страхования, номера кредитных карт и другие личные данные.
- **Tokens/Passwords** — обнаруживает токены API, пароли, ключи API, ключи доступа, секретные ключи, учётные данные аутентификации и другие конфиденциальные данные.
- **Jailbreak** — обнаруживает попытки обойти правила безопасности ИИ, манипулировать поведением модели или заставить её игнорировать свои инструкции.
- **Offensive Content** — обнаруживает оскорбительный, ненавистнический, дискриминационный, насильственный или неприемлемый контент.

- **Malicious Code** — обнаруживает потенциально вредоносный код, скрипты, эксплойты или опасные команды.
- **Prompt Injection** — обнаруживает попытки внедрения вредоносных инструкций, переопределения системных инструкций или манипулирования поведением ИИ через встроенные команды.

Если проверка проходит успешно, входные данные продолжают обработку через выход **Pass**. Если проверка не проходит, входные данные блокируются и отправляются через выход **Fail** с обоснованием причины отказа.

Ограничения **Jailbreak** и **Prompt Injection** включают дополнительное эвристическое обнаружение на первом этапе, а при необходимости переходят к проверке с помощью LLM. Этот дополнительный этап быстро выявляет очевидные шаблоны и снижает затраты на API, избегая ненужных вызовов LLM для явных нарушений.

Компонент **Ограничения** использует языковую модель для анализа входных данных и может генерировать ложные срабатывания или пропускать некоторые нарушения. Используйте этот компонент **в дополнение к** другим лучшим практикам защиты данных — например, обучению персонала и скриптам, проверяющим буквальные значения или шаблоны регулярных выражений — а не в качестве единственного средства защиты.

Использование компонента Ограничения в потоке

1. Подключите **Ввод чата** или другой источник текста к порту **Текстовый ввод** компонента **Ограничения**.
2. Выберите **Языковая модель**, который будет использоваться для проверки. Компонент использует подключённую LLM для анализа входного текста в соответствии с включёнными ограничениями.
3. В выпадающем списке **Ограничения** выберите одно или несколько ограничений для включения. Например, выберите **Токены/Пароли**, чтобы заблокировать ключи API и учётные данные.
4. Подключите выход **Pass** к компонентам, которые должны получать проверенные данные.
5. При необходимости подключите выход **Fail** для обработки заблокированных данных, например к компоненту **Вывод чата** или **Запись в файл**.

Создание пользовательских ограничений

Используйте параметр **Включить кастомные ограничения**, чтобы создать собственные специфические проверки. В поле **Объяснения кастомного ограничения** введите описание ограничения на естественном языке, описывающее недопустимые данные, которые вы хотите обнаружить.

Пользовательские ограничения работают одновременно со встроенными и следуют тому же процессу проверки.

Например, чтобы заблокировать ввод, содержащий упоминания названий конкурентов или их продуктов, введите следующее в поле **Описание пользовательского ограничения**:
competitor company names, competitor product names, or references to competing services

Когда это пользовательское ограничение включено, LLM анализирует входной текст согласно вашим критериям. Если обнаруживается контент, соответствующий описанию, проверка не проходит и ввод блокируется. В противном случае ввод продолжается через выход **Pass**.

Параметры компонента

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Языковая модель (model)	LanguageModel	Входной параметр. Подключите компонент Языковая модель для использования в качестве драйвера. Модель проверяет данные, сравнивает их с ограничениями и определяет наличие нарушений.
Ключ API (api_key)	Секретная строка	Входной параметр. Ключ API провайдера модели. Требуется, если провайдер требует аутентификации.
Ограничения (enabled_guardrails)	Множественный выбор	Входной параметр. Выберите одно или несколько ограничений для проверки ввода. Варианты: PII, Tokens/Passwords, Jailbreak, Offensive Content, Malicious Code, Prompt Injection. По умолчанию: ["PII", "Tokens/Passwords", "Jailbreak"].
Ввод текста (input_text)	Многострочная строка	Входной параметр. Текст для проверки на соответствие ограничениям. Принимает тип ввода Message.
Включить пользовательское ограничение (enable_custom_guardrail)	Булево	Входной параметр. Включает пользовательское ограничение с вашими собственными критериями проверки. По умолчанию: false.
Описание пользовательского ограничения (custom_guardrail_explanation)	Многострочная строка	Входной параметр. Описание того, что должно проверять пользовательское ограничение. Используется LLM для анализа ввода — будьте конкретны и ясны.

Название	Тип	Описание
Порог эвристического обнаружения (heuristic_threshold)	Ползунок	Применяется только когда enable_custom_guardrail равно true. Входной параметр. Пороговое значение (0.0–1.0) для эвристического обнаружения jailbreak и prompt injection. Сильные шаблоны, такие как «ignore instructions» и «jailbreak», имеют высокий вес; слабые шаблоны, такие как «bypass» и «act as», — низкий. Если суммарный балл достигает или превышает порог, ввод немедленно отклоняется. Более низкие значения — более строгая проверка; более высокие значения — больше случаев передаётся на проверку LLM. По умолчанию: 0.7.

25.6.3 Выбор LLM

До версии Генеративный ИИ 1.7 этот компонент назывался **LLM Router**. Компонент **Выбор LLM** направляет запросы к наиболее подходящей языковой модели на основе спецификаций моделей OpenRouter.

Чтобы использовать компонент в потоке, подключите несколько компонентов языковой модели к компоненту **Выбор LLM**. Одна модель выступает в роли судейской LLM: она анализирует входные сообщения, чтобы понять контекст задачи, выбирает наиболее подходящую модель из остальных подключённых LLM и направляет запрос к ней. Выбранная модель обрабатывает запрос и возвращает сгенерированный ответ.

В следующем примере потока три компонента языковой модели: один — судейская LLM, два других образуют пул моделей для маршрутизации запросов. Компоненты ввода и вывода создают бесшовное взаимодействие в чате: вы отправляете сообщение и получаете ответ, не замечая процесса маршрутизации.

Параметры компонента «Выбор LLM»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
models	Языковые модели	Входной параметр. Подключите вывод LanguageModel из

Название	Отображаемое название	Описание
		нескольких компонентов языковой модели для формирования пула моделей. <code>judge_llm</code> выбирает модель из этого пула при маршрутизации запросов. Первая подключённая модель используется по умолчанию при ошибках выбора или маршрутизации.
<code>input_value</code>	Ввод	Входной параметр. Входящий запрос, который будет направлен к модели, выбранной судейской LLM.
<code>judge_llm</code>	Модель-судья	Входной параметр. Подключите вывод <code>LanguageModel</code> из <i>одного</i> компонента языковой модели, чтобы использовать его в роли судейской LLM для маршрутизации запросов.
<code>optimization</code>	Оптимизация	Входной параметр. Задаёт предпочтительный критерий выбора модели. Варианты: <code>quality</code> (наивысшее качество ответа), <code>speed</code> (минимальное время отклика), <code>cost</code> (минимальная стоимость) или <code>balanced</code> (равный вес качества, скорости и стоимости). По умолчанию: <code>balanced</code> .
<code>use_openrouter_specs</code>	Использовать спецификации OpenRouter	Входной параметр. Определяет, нужно ли получать спецификации моделей из API OpenRouter. Если <code>false</code> , судейской LLM передаётся только название модели. По умолчанию: <code>true</code> .

Название	Отображаемое название	Описание
timeout	Тайм-аут API	Входной параметр. Тайм-аут в секундах для API-запросов, выполняемых маршрутизатором. По умолчанию: 10.
fallback_to_first	Резервная первая модель	Входной параметр. Определяет, нужно ли использовать первую модель из models в качестве резервной, если маршрутизация не смогла достичь выбранной модели. По умолчанию: true.

Выводы компонента «Выбор LLM»

Компонент **Выбор LLM** предоставляет три варианта вывода. Желаемый тип вывода задаётся рядом с выходным портом компонента.

- **Вывод** — объект Message с ответом на исходный запрос, сгенерированным выбранной LLM. Используйте этот вывод для обычных чат-взаимодействий.
- **Информация о выбранной модели** — объект Data со сведениями о выбранной модели: её названием и версией.
- **Решение о маршрутизации** — объект Message с обоснованием выбора судейской модели, включая длину входного запроса и количество рассмотренных моделей.

Это полезно для отладки, если вам кажется, что судейская модель не выбирает лучшую модель.

25.6.4 Умный маршрутизатор

Компонент **Умный маршрутизатор** — это вариант компонента **Если-иначе** на базе LLM. Вместо сопоставления строк **Умный маршрутизатор** использует подключённый компонент **Языковая модель** для категоризации и маршрутизации входящих сообщений.

Компонент **Умный маршрутизатор** можно использовать везде, где обычно применяют компонент **Если-иначе**. Для примера создайте пример потока с компонентом **Если-иначе**, а затем замените компонент **Если-иначе** на **Умный маршрутизатор**. Вместо регулярного выражения используйте таблицу **Маршруты** для определения выходных маршрутов сообщений.

Таблица **Маршруты** определяет категории для маршрутизации. Например, таблица маршрутов для анализа тональности может выглядеть так:

Название маршрута	Описание маршрута	Сообщение маршрута
Положительный	Положительная обратная связь, удовлетворение или комплименты	
Отрицательный	Жалобы, проблемы или неудовлетворённость	

Название маршрута	Описание маршрута	Сообщение маршрута
Нейтральный	Вопросы, запросы информации или нейтральные высказывания	Спасибо за ваш запрос!

Этот компонент создаёт порты для маршрутов **Положительный**, **Отрицательный** и **Нейтральный**. Когда LLM классифицирует входное сообщение, оно направляется к соответствующему выходному порту по имени маршрута. Для маршрутов **Положительный** и **Отрицательный** передаётся исходный входной текст. Для маршрута **Нейтральный** вместо исходного текста отправляется сообщение маршрута "Thank you for your inquiry!".

Параметр **Переопределить вывод** отправляет одно сообщение вне зависимости от того, какой маршрут выбрал LLM. Это сообщение имеет приоритет над всеми другими вариантами вывода и полностью заменяет как исходный входной текст, так и любые пользовательские сообщения маршрутов. В примере с анализом тональности, если установить **Переопределить вывод** в значение "Message received", все маршруты будут отправлять одно и то же сообщение.

Параметр **Дополнительные инструкции** добавляет дополнительные указания для LLM. Используйте заполнитель {input_text}, чтобы сослаться на входной текст для классификации, и {routes} — на список имён маршрутов, разделённых запятыми.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Языковая модель	LanguageModel	Входной параметр. Языковая модель для категоризации. LLM получает входной текст и доступные категории, затем возвращает точное название совпавшей категории. Обязательный.
Ввод	String	Входной параметр. Входной текст для категоризации. Обязательный.
Маршруты	Таблица	Входной параметр. Таблица, определяющая категории для маршрутизации. Каждая строка содержит имя маршрута (обязательно), необязательное описание, помогающее LLM понять категорию, и необязательное

Название	Тип	Описание
		пользовательское сообщение вывода. Компонент создаёт один выходной порт для каждой категории маршрута. Обязательный.
Переопределить вывод	Message	Входной параметр. Необязательное сообщение, которое имеет приоритет над всеми другими вариантами вывода. Если указано, заменяет как исходный входной текст, так и пользовательские сообщения маршрутов для всех маршрутов. Расширенный параметр.
Дополнительные инструкции	String	Входной параметр. Дополнительные инструкции для LLM-классификации. Добавляются к базовому запросу классификации, который уже включает полную таблицу Маршруты (имена и описания). Используйте {input_text} для входного текста и {routes} для списка имён маршрутов, разделённых запятыми.
Включить вывод Else	Boolean	Входной параметр. Включает выходной порт Else для случаев, когда ни один маршрут не совпал. Если отключено, вывод не производится при отсутствии совпадения. По умолчанию: false.
Else	Message	Выходной параметр. Доступен только когда Включить вывод Else равно true. Использует сообщение

Название	Тип	Описание
		переопределения (если указано) или исходный входной текст, если ни один маршрут не совпал.

25.6.5 Умное преобразование

Этот компонент был переименован несколько раз. Его предыдущие названия: **Lambda Filter** и **Smart Transform**.

Компонент **Умное преобразование** использует LLM и инструкции на естественном языке для создания лямбда-функции, которая фильтрует или преобразует входные данные типов Data, DataFrame или Message. Необходимо подключить этот компонент к компоненту языковой модели, который будет генерировать функцию на основе инструкций, указанных в параметре **Инструкции**. LLM выполняет функцию над входными данными и выводит результаты в виде Data, DataFrame или Message.

Давайте краткие и чёткие инструкции, сосредоточившись на желаемом результате или конкретных действиях, например: Filter the data to only include items where the 'status' is 'active'. Рекомендуется использовать одну фразу или меньше, поскольку знаки завершения предложения, такие как точка, могут вызывать ошибки или непредвиденное поведение.

Если необходимо предоставить более подробные инструкции, не относящиеся напрямую к лямбда-функции, их можно ввести в поле **Ввод** компонента **Языковая модель** или через компонент **Шаблон промпта**.

Для наиболее надёжных результатов тип вывода компонента **Умное преобразование** должен соответствовать типу входных данных. Например, выберите вывод **Сообщения** для входных данных типа Message.

В следующем примере компонент **API-запрос** передаёт JSON-данные из конечной точки <https://jsonplaceholder.typicode.com/users> в компонент **Умное преобразование**. Компонент **Умное преобразование** передаёт данные и инструкцию extract emails подключённому компоненту **Языковая модель**. LLM генерирует функцию-фильтр, извлекающую адреса электронной почты из JSON-данных, и возвращает отфильтрованные данные в виде сообщения чата.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
data	Данные	Входной параметр. Входные данные типа Data, DataFrame или Message для фильтрации или преобразования с помощью сгенерированной лямбда-функции.
model	Языковая модель	Входной параметр. Вывод LanguageModel из

Название	Отображаемое название	Описание компонента	Языковая модель.
filter_instruction	Инструкции	Входной параметр. Инструкции на естественном языке о том, как фильтровать или преобразовывать данные. LLM использует эти инструкции для создания лямбда-функции.	
sample_size	Размер выборки	Входной параметр. Для больших наборов данных — количество символов, которое следует отбирать из начала и конца набора данных. Применяется только если набор данных достигает или превышает значение max_size. По умолчанию: 1000.	
max_size	Максимальный размер	Входной параметр. Количество символов, при котором набор данных считается большим и активируется выборка по значению sample_size. По умолчанию: 30000.	

25.6.6 Структурированный вывод

Компонент **Структурированный вывод** использует LLM для преобразования произвольных входных данных в структурированные данные (Data или DataFrame), используя инструкции по форматированию на естественном языке и определение схемы вывода. Например, вы можете извлекать конкретные сведения из документов — электронных писем или научных статей.

Использование компонента «Структурированный вывод» в потоке

Чтобы использовать компонент **Структурированный вывод** в потоке:

1. Предоставьте **Входное сообщение** — исходный материал, из которого нужно извлечь структурированные данные. Обычно это данные из компонента **Ввод в чат**, **Чтение файла** или другого компонента, предоставляющего неструктурированный или полуструктурированный ввод.

Весь исходный материал не обязан превращаться в структурированный вывод. Сила компонента **Структурированный вывод** в том, что вы можете указать, какую именно информацию нужно извлечь, — даже если она явно не помечена или не соответствует точному ключевому слову. LLM проанализирует исходный материал, извлечёт релевантные данные и отформатирует их согласно вашим спецификациям. Нерелевантный материал в структурированный вывод не попадает.

2. Определите **Инструкции по форматированию** и **Схему вывода**, чтобы указать, какие данные следует извлечь и как их структурировать в итоговый объект Data или DataFrame.

Инструкции — это промпт, который сообщает LLM, какие данные извлекать, как их форматировать, как обрабатывать исключения и любые другие указания по подготовке структурированных данных.

Схема — это таблица, определяющая поля (ключи) и типы данных для организации извлечённых данных в структурированный объект Data или DataFrame. Для получения дополнительной информации см. Параметры схемы вывода.

2. Подключите компонент языковой модели, настроенный на вывод типа LanguageModel. LLM использует **Входное сообщение** и **Инструкции по форматированию** из компонента **Структурированный вывод**, чтобы извлечь конкретные фрагменты данных из входного текста. Схема вывода применяется к ответу модели для формирования итогового структурированного объекта Data или DataFrame.
3. Необязательно: как правило, структурированный вывод передаётся в последующие компоненты, которые используют извлечённые данные в дальнейших процессах, например в компоненты **Парсер** или **Операции с данными**.

Пример: шаблон Financial Report Parser

Шаблон **Financial Report Parser** демонстрирует, как компонент **Структурированный вывод** можно использовать для извлечения структурированных данных из неструктурированного текста.

Компонент **Структурированный вывод** в этом шаблоне настроен следующим образом:

- **Входное сообщение** поступает из компонента **Ввод в чат**, предварительно загруженного цитатами из образцов финансовых отчётов.
- **Инструкции по форматированию:**
You are an AI that extracts structured JSON objects from unstructured text.
Use a predefined schema with expected types (str, int, float, bool, dict).
Extract ALL relevant instances that match the schema - if multiple patterns exist, capture them all.
Fill missing or ambiguous values with defaults: null for missing values.
Remove exact duplicates but keep variations that have different field values.
Always return valid JSON in the expected format, never throw errors.
If multiple objects can be extracted, return them all in the structured format.
- **Схема вывода** включает ключи EBITDA, NET_INCOME и GROSS_PROFIT.

Структурированный объект Data передаётся в компонент **Парсер**, который формирует текстовую строку, подставляя ключи схемы в переменные шаблона разбора:

EBITDA: {EBITDA} , Net Income: {NET_INCOME} , GROSS_PROFIT: {GROSS_PROFIT}

При выводе в **Playground** итоговый Message подставляет вместо переменных фактические значения, извлечённые компонентом **Структурированный вывод**. Например:

EBITDA: 900 million , Net Income: 500 million , GROSS_PROFIT: 1.2 billion

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Языковая модель (llm)	LanguageModel	Входной параметр. Вывод LanguageModel из компонента Языковая модель , определяющий

Название	Тип	Описание
Входное сообщение (input_value)	String	LLM для анализа, извлечения и подготовки структурированного вывода. Входной параметр. Сообщение, содержащее исходный материал для извлечения.
Инструкции по форматированию (system_prompt)	String	Входной параметр. Инструкции для языковой модели по извлечению и форматированию вывода.
Имя схемы (schema_name)	String	Входной параметр. Необязательный заголовок для Схемы вывода .
Схема вывода (output_schema)	Таблица	Входной параметр. Таблица, описывающая схему желаемого структурированного вывода и определяющая содержимое объекта Data или DataFrame. См. Параметры схемы вывода.
Структурированный вывод (structured_output)	Data или DataFrame	Выходной параметр. Итоговый структурированный вывод, генерируемый компонентом. Рядом с выходным портом компонента можно выбрать тип данных вывода: Данные или DataFrame . Конкретное содержимое и структура вывода зависят от входных параметров.

Параметры схемы вывода

После того как LLM извлечёт релевантные данные из **Входного сообщения** согласно **Инструкциям по форматированию**, данные организуются в соответствии со **Схемой вывода**.

Схема — это таблица, определяющая поля (ключи) и типы данных для итогового вывода Data или DataFrame из компонента **Структурированный вывод**.

По умолчанию схема содержит одну строку field.

Чтобы добавить ключ в схему, нажмите **Добавить новую строку**, а затем заполните столбцы:

- **Название:** имя поля вывода. Как правило, это конкретный ключ, значение которого нужно извлечь. На эти ключи можно ссылаться как на переменные в последующих компонентах, например в шаблоне компонента **Парсер**. Например, ключ схемы `NET_INCOME` можно использовать в переменной `{NET_INCOME}`.
- **Описание:** необязательное описание содержимого поля и его назначения.
- **Тип:** тип данных значения, хранящегося в поле. Поддерживаемые типы: `str` (по умолчанию), `int`, `float`, `bool` и `dict`.
- **В виде списка:** включите эту настройку, если поле должно содержать список значений, а не одно значение.

Для простых схем можно ограничиться несколькими полями типа `str` или `int`. Для более сложных схем со списками и словарями полезно ознакомиться со структурами и атрибутами типов `Data` и `DataFrame`, описанными в разделе Типы данных Генеративный ИИ. Вы также можете сформировать черновой объект `Data` или `DataFrame`, а затем доработать его с помощью последующих компонентов, например **Операции с данными**.

25.7 Модели и агенты

25.7.1 Языковая модель

Компоненты языковой модели в модуле «Генеративный ИИ» генерируют текст с помощью указанной большой языковой модели (LLM). Они принимают входные данные — сообщения чата, файлы, инструкции — и генерируют текстовый ответ.

Генеративный ИИ включает основной компонент **Языковая модель** со встроенной поддержкой многих LLM. Вместо него можно использовать любую дополнительную языковую модель.

Использование компонентов языковой модели в потоках

Используйте компоненты языковой модели везде, где в потоке требуется LLM.

Один из самых распространённых сценариев — чат с LLM в ваших потоках.

Следующий пример использует компонент языковой модели в потоке чат-бота, аналогичном шаблону **Основной промпт**.

1. Чтобы настроить глобального провайдера моделей в модуле «Генеративный ИИ»:
 1. Нажмите значок профиля, выберите **Настройки**, затем нажмите **Провайдеры моделей**.
 2. На панели **Провайдеры моделей** выберите провайдера.
 3. В поле **Ключ API** добавьте API-ключ провайдера.
Ключ должен иметь разрешение на вызов нужных моделей, а на аккаунте должно быть достаточно кредитов.
Для каждого провайдера можно добавить только один ключ. Убедитесь, что ключ имеет доступ ко *всем* моделям, которые вы планируете использовать в модуле «Генеративный ИИ».
 4. Включите конкретные модели, которые вы хотите использовать. Доступные модели зависят от провайдера и разрешений ключа API. Модели, генерирующие текст, перечислены в разделе **Языковые модели**. Модели, генерирующие эмбединги, — в разделе **Модели эмбедингов**.

После включения модели в глобальной конфигурации её можно использовать в любом компоненте, управляемом моделью.

Если провайдер или модель не встроены в глобальные **Модели** Генеративный ИИ, замените компонент **Языковая модель** любым дополнительным компонентом языковой модели.

Посмотрите **Пакеты** или воспользуйтесь **поиском** по провайдеру.

Также можно использовать Ollama для локального размещения модели и настроить сервис Ollama в глобальных **Моделях** Генеративный ИИ. Либо создайте собственный пользовательский компонент с поддержкой нужного провайдера и модели — в качестве основы используйте существующий компонент языковой модели.

2. Добавьте основной компонент **Языковая модель** в поток и выберите модель в поле **Языковая модель**.

При необходимости нажмите **Управление провайдерами моделей**, чтобы открыть панель **Провайдеры моделей** и настроить API-ключи или включить/отключить модели.

2. В панели инспекции компонента включите параметр **Системное сообщение**.
3. Добавьте компонент **Шаблон промпта** в поток.
4. В поле **Шаблон** введите инструкции для LLM, например Вы — эксперт в области географии, занимающийся репетиторством со старшеклассниками.
5. Подключите выход компонента **Шаблон промпта** к входу **Системное сообщение** компонента **Языковая модель**.
6. Добавьте компоненты **Ввод в чат** и **Вывод чата** в поток. Они необходимы для прямого чат-взаимодействия с LLM.
7. Подключите компонент **Ввод в чат** к входу **Ввод** компонента **Языковая модель**, затем подключите выход **Сообщения** компонента **Языковая модель** к компоненту **Вывод чата**.
8. Откройте **Playground** и задайте вопрос, например Какая столица Швеции?.

Результат

Следующий ответ является примером ответа модели OpenAI. Реальный ответ может отличаться в зависимости от версии модели, шаблона и входных данных.

10. Необязательно: попробуйте другую модель или провайдера, чтобы сравнить ответы.

Если в глобальной панели **Провайдеры моделей** включено несколько моделей, выберите другую в поле **Языковая модель**. Чтобы открыть панель, нажмите значок профиля, выберите **Настройки**, затем **Провайдеры моделей**. Откройте **Playground**, задайте тот же вопрос и сравните содержание и формат ответов.

Это помогает понять, как разные модели обрабатывают один запрос, и выбрать оптимальную модель для вашего сценария. Подробнее о конкретных моделях см. в документации соответствующего провайдера.

Некоторые компоненты используют компонент языковой модели для выполнения действий, управляемых LLM. Как правило, такие компоненты подготавливают данные для дальнейшей обработки последующими компонентами, а не выводят прямой чат-ответ. Пример см. в разделе Компонент **Умное преобразование**.

Компонент должен принимать вход LanguageModel, а тип вывода компонента языковой модели должен быть установлен в LanguageModel. Подробнее см. в разделе Типы вывода языковой модели.

Если вы не хотите использовать встроенную LLM компонента **Агент**, подключите предпочитаемую модель через компонент языковой модели:

1. Добавьте компонент языковой модели в поток.

Можно использовать основной компонент **Языковая модель** или найти дополнительные варианты в **Пакетах**. Компоненты в пакетах могут не содержать слов language model в названии. Например, LLM Azure OpenAI доступны через компонент **Azure OpenAI**.

2. Выберите предпочитаемую модель в выпадающем списке **Языковая модель**. Модель должна быть настроена глобально в панели **Модели**.
3. Измените тип вывода компонента языковой модели с **Ответ модели** на **Языковая модель**. Порт вывода изменится на порт LanguageModel. Это необходимо для подключения компонента языковой модели к компоненту **Агент**. Подробнее см. в разделе Типы вывода языковой модели.
4. Добавьте компонент **Агент** в поток.
5. Подключите выход компонента языковой модели к входу **Языковая модель** компонента **Агент**. Компонент **Агент** теперь будет использовать настройки подключённой языковой модели вместо встроенных.

Параметры языковой модели

Следующие параметры относятся к основному компоненту **Языковая модель**. У других компонентов языковой модели могут быть дополнительные или отличающиеся параметры.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Изменить все параметры можно через панель инспекции компонента, которая открывается при выборе компонента.

Параметр	Тип	Описание
provider	String	Входной параметр. Провайдер модели. Варианты зависят от вашей глобальной конфигурации Модели .
model_name	String	Входной параметр. Название модели. Варианты зависят от выбранного провайдера и глобальной конфигурации Модели .
input_value	String	Входной параметр. Текст для отправки модели.
system_message	String	Входной параметр. Системное сообщение, задающее поведение ассистента.
stream	Boolean	Входной параметр. Поточковая передача ответа. По умолчанию: false.
temperature	Float	Входной параметр. Управляет случайностью ответов. Диапазон: [0.0, 1.0]. По умолчанию: 0.1.
model	LanguageModel	Выходной параметр. Альтернативный тип

Параметр	Тип	Описание
		вывода вместо Message по умолчанию. Создаёт экземпляр чата, настроенный с указанными параметрами. См. Типы вывода языковой модели.

Типы вывода языковой модели

Компоненты языковой модели — как основной компонент, так и компоненты из пакетов — могут генерировать два типа вывода:

- **Ответ модели:** тип вывода по умолчанию. Возвращает сгенерированный ответ в виде данных Message. Используйте этот тип, когда нужен стандартный диалог с LLM — LLM генерирует текстовый ответ на основе переданного ввода.
- **Языковая модель:** измените тип вывода на LanguageModel, когда необходимо подключить LLM к другому компоненту в потоке, например **Агент** или **Умное преобразование**.

В этой конфигурации компонент языковой модели обеспечивает работу другого компонента, а не прямое чат-взаимодействие. Например, компонент **Умное преобразование** использует LLM для создания функции из описания на естественном языке.

Дополнительные языковые модели

Если ваш провайдер или модель не поддерживается основным компонентом **Языковая модель**, дополнительные компоненты доступны в разделе **Пакеты**.

Их можно использовать так же, как основной компонент **Языковая модель**, как описано в разделе **Использование компонентов языковой модели в потоках**.

Языковые модели и векторные хранилища

Векторные данные — неотъемлемая часть приложений на основе LLM, таких как чат-боты и агенты.

LLM можно использовать самостоятельно для обычных чат-взаимодействий и стандартных задач, однако контекстно-зависимые приложения (например, RAG) и работа с пользовательскими наборами данных (например, внутренними бизнес-данными) требуют интеграции векторных баз данных. Они предоставляют дополнительный контекст и формируют осмысленные запросы.

Генеративный ИИ включает компоненты векторных хранилищ для чтения и записи векторных данных: хранение эмбеддингов, поиск по сходству, Graph RAG и специализированные поисковые компоненты, например OpenSearch. Из-за их взаимозависимой функциональности компоненты векторного хранилища, языковой модели и модели эмбеддингов обычно используются в одном потоке или серии связанных потоков.

Чтобы найти доступные компоненты векторных хранилищ, просмотрите **Пакеты** или воспользуйтесь **поиском** по провайдеру.

Пример: поток векторного поиска

Руководство по использованию векторных данных в потоке см. в разделе **Создание чат-бота с векторным RAG**. Следующий пример демонстрирует использование компонентов векторного хранилища совместно с моделями эмбеддингов и языковыми моделями. Описаны важные детали конфигурации, функциональность и лучшие практики. Это лишь один из возможных сценариев — не исчерпывающее руководство для всех конфигураций.

1. Создайте поток на основе шаблона **Векторное хранилище RAG**.

Шаблон содержит два подпотока. Подпоток **Загрузка данных** загружает эмбединги и контент в векторную базу данных, а подпоток **Поисковый модуль** выполняет векторный поиск для получения релевантного контекста по запросу пользователя.

2. Настройте соединение с базой данных для обоих компонентов **Astra DB** или замените их другой парой компонентов векторного хранилища. Убедитесь, что оба компонента подключаются к одному и тому же векторному хранилищу, и что компонент в подпотоке **Поисковый модуль** может выполнять поиск по сходству.

Параметры каждого компонента векторного хранилища зависят от его роли в потоке. В этом примере подпоток **Загрузка данных** *записывает* в векторное хранилище, а подпоток **Поисковый модуль** *читает* из него. Параметры, связанные с поиском, актуальны только для компонента **Векторный поиск** в подпотоке **Поисковый модуль**.

Описание конкретных параметров см. в документации выбранного компонента векторного хранилища.

3. Настройте модель эмбедингов одним из следующих способов:

- **Использовать модель OpenAI:** в обоих компонентах **OpenAI Эмбединги** введите ваш API-ключ OpenAI. Можно использовать модель по умолчанию или выбрать другую.
- **Использовать другого провайдера:** замените компоненты **OpenAI Эмбединги** другой парой компонентов модели эмбедингов и настройте параметры и учётные данные.
- **Использовать векторизацию Astra DB:** если вы используете векторное хранилище Astra DB с интеграцией векторизации, оба компонента **OpenAI Эмбединги** можно удалить. В этом случае интеграция автоматически генерирует эмбединги из **Загружаемых данных** и **Поискового запроса**.

Если векторное хранилище уже содержит эмбединги, убедитесь, что компоненты модели эмбедингов используют ту же модель, что и при их создании. Смешивание моделей эмбедингов в одном хранилище может привести к неточным результатам поиска. 4. Рекомендация: в компоненте **Разделить текст** оптимизируйте параметры разбиения для вашей модели эмбедингов. Например, если у модели эмбедингов ограничение в 512 токенов, параметр **Размер фрагмента** не должен превышать это значение.

Поскольку подпоток **Поисковый модуль** передаёт ввод чата напрямую в компонент векторного хранилища, убедитесь, что строка ввода не превышает ограничения модели эмбедингов. В тестовом сценарии можно просто вводить запросы в пределах ограничений; в рабочей среде может потребоваться реализовать дополнительные проверки или шаги предобработки. Например, добавьте компоненты для подготовки ввода чата или ограничьте его длину в коде приложения.

5. В компоненте **Языковая модель** введите API-ключ OpenAI или выберите другого провайдера и модель для чат-части потока.

6. Запустите подпоток **Загрузка данных**, чтобы заполнить векторное хранилище. В компоненте **Чтение файла** выберите один или несколько файлов, затем нажмите **Запустить компонент** на компоненте векторного хранилища в подпотоке **Загрузка данных**.

Подпоток загружает файлы с локального компьютера, разбивает их на части, генерирует эмбединги для каждой части и сохраняет части с эмбедингами в векторной базе данных.

Подпоток **Загрузка данных** отделён от подпотока **Поисковый модуль**, поскольку предзагрузку данных, как правило, не нужно выполнять при каждом чат-запросе. Запускайте его по мере необходимости для предзагрузки или обновления данных в хранилище. Если хранилище уже содержит нужные данные, запускать подпоток **Загрузка данных** не требуется.

7. Откройте **Playground** и начните чат, чтобы запустить подпоток **Поисковый модуль**.

Подпоток генерирует эмбединг из ввода чата, выполняет векторный поиск для получения похожего контента из хранилища, обрабатывает результаты поиска в дополнительный контекст для LLM, после чего LLM генерирует ответ на естественном языке. LLM использует результаты векторного поиска совместно с внутренними обучающими данными и инструментами, такими как веб-поиск и информация о дате/времени.

Чтобы избежать передачи полного блока необработанных результатов поиска, компонент **Парсер** извлекает строки text из объекта результатов Data и передаёт их в компонент **Шаблон промпта** в формате Message. Там строки и содержимое шаблона компилируются в инструкции для LLM.

Для этого преобразования можно использовать и другие компоненты, например **Операции с данными**, — в зависимости от того, как вы хотите использовать результаты поиска.

Чтобы просмотреть необработанные результаты поиска, нажмите **Проверить вывод** на компоненте векторного хранилища после запуска подпотока **Поисковый модуль**.

25.7.2 Шаблон промпта

Используйте основной компонент **Шаблон промпта**, чтобы создать промпт с инструкциями и контекстом для LLM или агента — отдельно от других входных данных, таких как сообщения чата или загрузка файлов.

Промпты — это структурированный ввод, использующий естественный язык, фиксированные значения и динамические переменные для передачи базового контекста LLM. Например:

- Задайте единообразную структуру запросов пользователей, чтобы LLM легче понимала их и корректно отвечала.
- Задайте конкретный формат вывода для LLM, например JSON или структурированный текст.
- Определите роль LLM, например Ты эксперт или Ты помощник.
- Предоставьте LLM доступ к истории чата.

Компонент **Шаблон промпта** также может передавать переменные-инструкции другим компонентам в потоке.

Параметры компонента

Параметр	Отображаемое название	Описание
template	Шаблон	Входной параметр. Создайте шаблон промпта с динамическими переменными в фигурных скобках, например {VARIABLE_NAME}. Если шаблон содержит буквальный текст и переменные, используйте

Параметр	Отображаемое название	Описание
		двойные фигурные скобки для экранирования буквального текста, чтобы он не интерпретировался как переменная. Например: This is a template with {{literal text in curly braces}} and a {variable}. Если шаблон содержит много буквальных фигурных скобок, например структуры JSON, рассмотрите использование шаблонизации Mustache. Подробнее см. в разделе Шаблонизация Mustache.
use_double_brackets	Использовать двойные скобки	Когда включено, используется синтаксис Mustache {{variable}} вместо синтаксиса f-строк {variable}. Подробнее см. в разделе Шаблонизация Mustache.

Определение переменных в промпте

Переменные в компоненте **Шаблон промпта** динамически добавляют поля в компонент, благодаря чему поток может получать значения из других компонентов, глобальных переменных Генеративный ИИ или фиксированного ввода.

Например, с помощью компонента **История сообщений** вы можете использовать переменную {memory} для передачи истории чата в промпт. Однако компонент **Агент** включает встроенную память чата, включённую по умолчанию. Подробнее см. в разделе Параметры управления памятью.

Следующие шаги демонстрируют добавление переменных в компонент **Шаблон промпта**:

1. Создайте поток на основе шаблона **Основной промпт**.

Шаблон уже содержит компонент **Шаблон промпта**, но содержит только инструкции на естественном языке.

2. Щёлкните компонент **Шаблон промпта** и добавьте переменные в поле **Шаблон**.

Переменные объявляются путём заключения имени переменной в фигурные скобки, например {variable_name}. Следующий шаблон создаёт переменные context и user_question: Given the context:

```
{context}
```

Answer the question:

```
{user_question}
```


Если шаблон содержит буквальный текст и переменные, используйте двойные фигурные скобки для экранирования буквального текста. Например: This is a template with `{{literal text in curly braces}}` and a `{variable}`.

Если шаблон содержит много буквальных фигурных скобок, например структуры JSON, рассмотрите использование шаблонизации Mustache. Подробнее см. в разделе Шаблонизация Mustache.

3. Нажмите **Проверить и сохранить**.

После добавления переменных в шаблон в компоненте **Шаблон промпта** появятся новые поля для каждой переменной.

4. Задайте значения для полей переменных:

- Подключите поля к другим компонентам, чтобы передавать их вывод в переменные.
- Используйте глобальные переменные Генеративный ИИ.
- Введите фиксированные значения непосредственно в поля.

В шаблон можно добавить любое количество переменных. Например, можно добавить переменные `{references}` и `{instructions}` и передавать в них данные из компонентов **Ввод текста**, **URL** или **Чтение файла**.

Шаблонизация Mustache в промтах

Экранирование f-строк может быть неудобным, когда в одном шаблоне смешиваются экранированные скобки и переменные. Например:

Generate a response in this JSON format:

```
{{"name": "{name}", "age": {age}, "city": "{city}"}}
```

The user's name is {name}, age is {age}, and they live in {city}.

Здесь `{{` и `}}` — это экранированные буквальные скобки для структуры JSON, а `{name}` — переменная. Такой синтаксис может приводить к ошибкам и затруднять чтение шаблона. Используйте Mustache, чтобы сделать различие более очевидным.

Чтобы включить шаблонизацию Mustache:

1. В компоненте **Шаблон промпта** включите **Использовать двойные скобки**.
2. В шаблоне промпта замените `{variable}` на `{{variable}}`. В Mustache `{ }` используются для буквальных скобок, а `{{variable}}` — для переменных.

Generate a response in this JSON format:

```
{{"name": "{{name}}", "age": {{age}}, "city": "{{city}}"}}
```

The user's name is {{name}}, age is {{age}}, and they live in {{city}}.

3. Нажмите **Проверить и сохранить**. Компонент проверит шаблон и вернёт **Запрос готов**, если ошибок нет.

Генеративный ИИ поддерживает подстановку переменных в синтаксисе Mustache, но не реализует полный движок Mustache. Валидатор компонента отклоняет синтаксис других возможностей Mustache, таких как циклы и условные конструкции.

25.7.3 Агенты

Компонент **Агент** в модуле «Генеративный ИИ» является ключевым для построения агентных потоков. Этот компонент определяет поведение и возможности AI-агентов в ваших потоках.

Как работают агенты?

Агенты расширяют большие языковые модели (LLM), интегрируя *инструменты* — функции, предоставляющие дополнительный контекст и позволяющие автономно выполнять задачи. Эти интеграции делают агентов более специализированными и мощными по сравнению с отдельными LLM.

Если LLM генерирует приемлемые, но пассивные ответы на общие запросы, то агент использует контекст и инструменты, чтобы давать более релевантные ответы и предпринимать конкретные действия. Например, можно создать агента с доступом к документации, репозиториям и другим ресурсам компании, чтобы он помогал команде решать задачи, требующие знания конкретных продуктов, клиентов и кода.

Агент использует LLM как движок рассуждений: обрабатывает входные данные, определяет необходимые действия и генерирует ответ. Ответ может представлять собой обычный текстовый ответ LLM или включать действие — редактирование файла, запуск скрипта или вызов внешнего API.

В контексте агентов инструменты — это функции, которые агент выполняет для решения задач или доступа к внешним ресурсам. Функция оборачивается в объект Tool с единым интерфейсом, понятным агенту. Агент узнаёт об инструментах через их регистрацию: при инициализации агенту предоставляется список доступных инструментов. Описание объекта Tool сообщает агенту, что умеет инструмент, — это позволяет агенту решить, подходит ли инструмент для конкретного запроса.

Примеры агентных потоков

Примеры потоков с использованием компонента **Агент**:

- Быстрый старт Генеративный ИИ: начните с шаблона **Простой агент**, измените его инструменты, а затем изучите, как использовать агентный поток в приложении. Шаблон **Простой агент** создаёт базовый агентный поток с компонентом **Агент**, который может использовать два других компонента Генеративный ИИ в качестве инструментов. LLM, указанный в настройках компонента **Агент**, может применять как встроенные функции, так и функции подключённых инструментов при генерации ответов.
- Использовать агента как инструмент: создайте мультиагентный поток.
- Использовать Генеративный ИИ как MCP-клиент и как MCP-сервер: используйте компонент **Агент** и компонент **Инструменты MCP** для реализации протокола Model Context Protocol (MCP) в ваших потоках.

Компонент Агент

Компонент **Агент** является основой агентных потоков. Он использует интеграцию с LLM для обработки входных данных — сообщений чата, загрузки файлов и других источников.

Агент может использовать инструменты, уже встроенные в базовый LLM, а также дополнительные инструменты, подключённые к порту **Инструменты** компонента **Агент**. К нему можно подключить любой компонент Генеративный ИИ, включая другие компоненты **Агент** и MCP-серверы через компонент **Инструменты MCP**.

Подробнее об использовании компонента см. в разделе **Агенты Генеративный ИИ**.

25.7.4 Инструменты MCP

Компонент **Инструменты MCP** подключается к серверу MCP (Model Context Protocol) и предоставляет функции этого сервера в виде инструментов для использования агентами Генеративный ИИ при обработке входных данных.

Помимо общедоступных серверов MCP и ваших собственных серверов MCP, вы можете подключать серверы MCP Генеративный ИИ, которые позволяют агенту использовать ваши потоки Генеративный ИИ в качестве инструментов. Для этого используйте режим HTTP/SSE компонента **Инструменты MCP** для подключения к серверу MCP вашего проекта Генеративный ИИ.

Для получения дополнительной информации см. Генеративный ИИ в роли клиента MCP и Генеративный ИИ в роли сервера MCP.

Если вы обновляете существующий поток и компонент **Инструменты MCP** теряет опцию **Режим инструмента**, см. Компонент «Инструменты MCP» теряет опцию «Режим инструмента» после обновления потоков.

Параметры компонента

Название	Тип	Описание
mcp_server	String	Входной параметр. Сервер MCP для подключения. Выберите из ранее настроенных серверов или добавьте новый.
tool	String	Входной параметр. Конкретный инструмент для выполнения на подключённом сервере MCP. Оставьте пустым, чтобы предоставить доступ ко всем инструментам.
use_cache	Boolean	Входной параметр. Включить кэширование сервера MCP и инструментов для повышения производительности. По умолчанию: false.
verify_ssl	Boolean	Входной параметр. Включить проверку SSL-сертификата для HTTPS-соединений. По умолчанию: true.
response	DataFrame	Выходной параметр. Объект DataFrame, содержащий ответ от выполненного инструмента.

Ранние версии компонента «Инструменты MCP»

- В версии Генеративный ИИ 1.5 компонент **MCP Connection** был переименован в компонент **Инструменты MCP**.
- В версии Генеративный ИИ 1.3 компоненты **Инструменты MCP (stdio)** и **Инструменты MCP (SSE)** были удалены и заменены единым компонентом **MCP Connection**, который позднее был переименован в **Инструменты MCP**.

25.7.5 Модель эмбедингов

Компоненты модели эмбедингов в модуле «Генеративный ИИ» генерируют текстовые эмбединги с помощью указанной большой языковой модели (LLM).

Генеративный ИИ включает основной компонент **Модель эмбедингов** со встроенной поддержкой ряда LLM.

Вместо него можно использовать любую дополнительную модель эмбедингов.

Использование компонентов модели эмбедингов в потоке

Используйте компоненты модели эмбедингов везде, где в потоке требуется генерация эмбедингов.

В этом примере показано, как использовать компонент модели эмбедингов для создания системы семантического поиска. Поток загружает текстовый файл, разбивает текст на фрагменты, генерирует эмбединги для каждого фрагмента и загружает фрагменты с эмбедингами в векторное хранилище. Компоненты ввода и вывода позволяют пользователю выполнять запросы к хранилищу через чат-интерфейс.

1. Создайте поток, добавьте компонент **Чтение файла** и выберите текстовый файл для тестирования, например PDF.
2. Настройте глобального провайдера моделей в модуле «Генеративный ИИ»:

1. Нажмите значок профиля, выберите **Настройки**, затем нажмите **Провайдеры моделей**.

2. На панели **Провайдеры моделей** выберите провайдера.

3. В поле **Ключ API** добавьте API-ключ провайдера.

Ключ должен иметь разрешение на вызов нужных моделей, а на аккаунте должно быть достаточно кредитов.

Для каждого провайдера можно добавить только один ключ. Убедитесь, что ключ имеет доступ ко *всем* моделям, которые вы планируете использовать в модуле «Генеративный ИИ».

4. Включите конкретные модели, которые вы хотите использовать.

Доступные модели зависят от провайдера и разрешений ключа API.

Модели, генерирующие текст, перечислены в разделе **Языковые модели**.

Модели, генерирующие эмбединги, — в разделе **Модели эмбедингов**.

После включения модели в глобальной конфигурации её можно использовать в любом компоненте, управляемом моделью.

Если провайдер или модель не доступны в глобальном разделе **Модели** Генеративный ИИ, используйте любую дополнительную модель эмбедингов вместо основного компонента.

Просмотрите **Пакеты** или воспользуйтесь **поиском** по провайдеру, например найдите компонент **Вывод эмбедингов Hugging Face**.

3. Добавьте основной компонент **Модель эмбедингов** в поток и выберите настроенную модель в выпадающем списке **Модель эмбедингов**.
3. Добавьте компонент **Разделить текст**. Он разбивает входной текст на фрагменты для последующего преобразования в эмбединги.
4. Добавьте компонент векторного хранилища, например **Chroma DB**, и настройте подключение к вашему хранилищу. Компонент сохраняет сгенерированные эмбединги для поиска по сходству.
5. Подключите компоненты:
 - Выход **Загруженные файлы** компонента **Чтение файла** → вход **Данные** или **DataFrame** компонента **Разделить текст**.
 - Выход **Чанки** компонента **Разделить текст** → вход **Загружаемые данные** компонента векторного хранилища.

- Выход **Эмбединги** компонента **Модель эмбедингов** → вход **Эмбединги** компонента векторного хранилища.
- 6. Чтобы выполнять запросы к векторному хранилищу, добавьте компоненты **Ввод в чат** и **Вывод чата**:
 - Выход компонента **Ввод в чат** → вход **Поисковый запрос** компонента векторного хранилища.
 - Выход **Результаты поиска** компонента векторного хранилища → компонент **Вывод чата**.
- 7. Нажмите **Playground** и введите поисковый запрос, чтобы получить фрагменты текста, наиболее семантически близкие к запросу.

Параметры модели эмбедингов

Следующие параметры относятся к основному компоненту **Модель эмбедингов**. Другие компоненты модели эмбедингов могут иметь дополнительные или отличающиеся параметры.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Изменить все параметры можно через панель инспекции компонента, которая открывается при выборе компонента.

Параметр	Отображаемое название	Тип	Описание
provider	Провайдер модели	List	Входной параметр. Выберите провайдера модели эмбедингов. Модели настраиваются глобально в панели Модели .
model	Название модели	List	Входной параметр. Выберите модель эмбедингов. Варианты зависят от выбранного провайдера и глобальной конфигурации Модели .
api_base	Базовый URL API	String	Входной параметр. Базовый URL для API. Оставьте пустым для значения по умолчанию.
dimensions	Размерность	Integer	Входной параметр.

Параметр	Отображаемое название	Тип	Описание
chunk_size	Размер фрагмента	Integer	Количество измерений выходных эмбеддингов. Входной параметр. Размер текстовых фрагментов для обработки. По умолчанию: 1000.
request_timeout	Тайм-аут запроса	Float	Входной параметр. Тайм-аут для запросов к API.
max_retries	Макс. число повторов	Integer	Входной параметр. Максимальное количество повторных запросов. По умолчанию: 3.
show_progress_bar	Индикатор выполнения	Boolean	Входной параметр. Отображать ли индикатор прогресса при генерации эмбеддингов.
model_kwargs	Параметры модели	Dictionary	Входной параметр. Дополнительные именованные аргументы для модели.
embeddings	Эмбеддинги	Embeddings	Выходной параметр. Экземпляр для генерации эмбеддингов с использованием выбранного провайдера.

Дополнительные модели эмбеддингов

Если ваш провайдер или модель недоступны в глобальном разделе **Модели** Генеративный ИИ, замените основной компонент **Модель эмбедингов** любым другим компонентом, генерирующим эмбединги.

Чтобы найти дополнительные компоненты модели эмбедингов, просмотрите **Пакеты** или воспользуйтесь **поиском** по провайдеру.

Модели эмбедингов и векторные хранилища

Векторные данные — неотъемлемая часть LLM-приложений, таких как чат-боты и агенты.

LLM можно использовать самостоятельно для обычных чат-взаимодействий и стандартных задач, однако контекстно-зависимые приложения (например, RAG) и работа с пользовательскими наборами данных (например, внутренними бизнес-данными) требуют интеграции векторных баз данных. Они предоставляют дополнительный контекст и формируют осмысленные запросы.

Генеративный ИИ включает компоненты векторных хранилищ для чтения и записи векторных данных: хранение эмбедингов, поиск по сходству, Graph RAG и специализированные поисковые компоненты, например OpenSearch. Из-за их взаимозависимой функциональности компоненты векторного хранилища, языковой модели и модели эмбедингов обычно используются в одном потоке или серии связанных потоков.

Чтобы найти доступные компоненты векторных хранилищ, просмотрите **Пакеты** или воспользуйтесь **поиском** по провайдеру.

Пример: поток векторного поиска

Руководство по использованию векторных данных в потоке см. в разделе Создание чат-бота с векторным RAG. Следующий пример демонстрирует использование компонентов векторного хранилища совместно с моделями эмбедингов и языковыми моделями. Описаны важные детали конфигурации, функциональность и лучшие практики. Это лишь один из возможных сценариев — не исчерпывающее руководство для всех конфигураций.

1. Создайте поток на основе шаблона **Векторное хранилище RAG**.

Шаблон содержит два подпотока. Подпоток **Загрузка данных** загружает эмбединги и контент в векторную базу данных, а подпоток **Поисковый модуль** выполняет векторный поиск для получения релевантного контекста по запросу пользователя.

2. Настройте соединение с базой данных для обоих компонентов **Astra DB**, либо замените их другой парой компонентов векторного хранилища. Убедитесь, что оба компонента подключены к одному и тому же векторному хранилищу, и что компонент в подпотоке **Поисковый модуль** может выполнять поиск по сходству.

Параметры каждого компонента зависят от его роли в потоке. В этом примере подпоток **Загрузка данных** *записывает* в векторное хранилище, а подпоток **Поисковый модуль** *читает* из него. Поэтому параметры поиска актуальны только для компонента **Векторный поиск** в подпотоке **Поисковый модуль**.

Описание конкретных параметров см. в документации выбранного компонента векторного хранилища.

3. Настройте модель эмбедингов одним из следующих способов:

- **Использовать модель OpenAI:** в обоих компонентах **OpenAI Эмбединги** введите API-ключ OpenAI. Можно использовать модель по умолчанию или выбрать другую.
- **Использовать другого провайдера:** замените компоненты **OpenAI Эмбединги** другой парой компонентов модели эмбедингов и настройте параметры и учётные данные.

- **Использовать векторизацию Astra DB:** если вы используете векторное хранилище Astra DB с интеграцией векторизации, оба компонента **OpenAI Эмбединги** можно удалить. В этом случае интеграция автоматически генерирует эмбединги из **Загружаемых данных** и **Поискового запроса**.

Если векторное хранилище уже содержит эмбединги, убедитесь, что компоненты модели эмбедингов используют ту же модель, что и при их создании. Смешивание моделей эмбедингов в одном хранилище может привести к неточным результатам поиска. 4. Рекомендация: в компоненте **Разделить текст** оптимизируйте параметры разбиения для вашей модели эмбедингов. Например, если у модели ограничение в 512 токенов, параметр **Размер фрагмента** не должен превышать это значение.

Поскольку подпоток **Поисковый модуль** передаёт ввод чата напрямую в компонент векторного хранилища, убедитесь, что строка ввода не превышает ограничения модели эмбедингов. В тестовом сценарии достаточно вводить запросы в пределах ограничений; в рабочей среде может потребоваться реализовать дополнительные проверки или шаги предобработки. Например, добавьте компоненты для подготовки ввода чата или ограничьте его длину в коде приложения.

5. В компоненте **Языковая модель** введите API-ключ OpenAI или выберите другого провайдера и модель для чат-части потока.
6. Запустите подпоток **Загрузка данных**, чтобы заполнить векторное хранилище. В компоненте **Чтение файла** выберите один или несколько файлов, затем нажмите **Запустить компонент** на компоненте векторного хранилища в подпотоке **Загрузка данных**.

Подпоток загружает файлы с локального компьютера, разбивает их на части, генерирует эмбединги для каждой части и сохраняет части с эмбедингами в векторной базе данных.

Подпоток **Загрузка данных** отделён от подпотока **Поисковый модуль**, поскольку предзагрузку данных не нужно выполнять при каждом чат-запросе. Запускайте его по мере необходимости для предзагрузки или обновления данных в хранилище. Если хранилище уже содержит нужные данные, запускать подпоток **Загрузка данных** не требуется.

7. Откройте **Playground** и начните чат, чтобы запустить подпоток **Поисковый модуль**.

Подпоток генерирует эмбединг из ввода чата, выполняет векторный поиск для получения похожего контента из хранилища, обрабатывает результаты поиска в дополнительный контекст для LLM, после чего LLM генерирует ответ на естественном языке. LLM использует результаты векторного поиска совместно с внутренними обучающими данными и инструментами, такими как веб-поиск и информация о дате/времени.

Чтобы избежать передачи полного блока необработанных результатов поиска, компонент **Парсер** извлекает строки text из объекта результатов Data и передаёт их компоненту **Шаблон промпта** в формате Message. Там строки и содержимое шаблона компилируются в инструкции для LLM.

Для этого преобразования можно использовать и другие компоненты, например **Операции с данными**, — в зависимости от того, как вы хотите использовать результаты поиска.

Чтобы просмотреть необработанные результаты поиска, нажмите **Проверить вывод** на компоненте векторного хранилища после запуска подпотока **Поисковый модуль**.

25.7.6 История сообщений

Компонент **История сообщений** предоставляет объединённую функциональность истории чата и хранения сообщений. Он может сохранять и извлекать сообщения чата из

хранилища Генеративный ИИ или из выделенной базы данных памяти чата, такой как Mem0 или Redis.

Компонент **Агент** имеет встроенную память чата, которая включена по умолчанию и использует хранилище Генеративный ИИ. Этой встроенной функциональности достаточно для большинства сценариев использования.

Используйте компонент **История сообщений** в следующих случаях:

- Необходимо хранить и извлекать память чата для компонента языковой модели (не агента).
- Нужно извлекать память чата вне контекста чата — например, в потоке анализа тональности, который извлекает и анализирует недавно сохранённые данные.
- Требуется хранить данные в конкретной базе данных, отдельной от хранилища Генеративный ИИ.

Для получения дополнительной информации см. [Хранение памяти чата](#).

Использование компонента в потоке

Компонент **История сообщений** работает в двух режимах в зависимости от его места в потоке:

- **Режим получения** — компонент извлекает сообщения чата из базы данных Генеративный ИИ или внешней памяти.
- **Режим сохранения** — компонент сохраняет сообщения чата в базу данных Генеративный ИИ или внешнюю память.

Если нужно одновременно сохранять и извлекать сообщения чата, используйте два отдельных компонента **История сообщений** в потоке.

Следующие шаги описывают создание потока на основе чата, использующего компоненты **История сообщений** для сохранения и извлечения памяти чата из базы данных Генеративный ИИ:

1. Создайте или откройте поток, в котором хотите использовать память чата.
2. В начале потока добавьте компонент **История сообщений** и установите его в режим **Retrieve**.
3. Необязательно: нажмите на компонент **История сообщений**, чтобы открыть панель инспекции компонента и настроить параметры сортировки, фильтрации и ограничения памяти.
4. Добавьте компонент **Шаблон промпта**, укажите переменную `{memory}` в поле **Шаблон**, затем подключите выход **История сообщений** к входу **memory**.

Компонент **Шаблон промпта** передаёт инструкции и контекст в языковую модель отдельно от сообщений чата, поступающих через компонент **Ввод в чат**. Шаблон может содержать любой текст и переменные, например:

You are a helpful assistant that answers questions.

Use markdown to format your answer, properly embedding images and urls.

History:

`{memory}`

Переменные в формате `{variable}` динамически создают поля в компоненте **Шаблон промпта**, через которые поток получает значения из других компонентов, глобальных переменных Генеративный ИИ или ввода во время выполнения. Подробнее см. [Определение переменных в промптах](#).

В этом примере переменная {memory} заполняется извлечённой памятью чата, которая затем передаётся в компонент **Языковая модель** или **Агент** как дополнительный контекст.

5. Подключите выход компонента **Шаблон промпта** к входу **Системное сообщение** компонента **Языковая модель**.

В этом примере центральным драйвером чата служит компонент **Языковая модель**, но можно использовать другой компонент языковой модели или компонент **Агент**.

6. Добавьте компонент **Ввод в чат** и подключите его к входу **Ввод** компонента **Языковая модель**.

7. Подключите выход компонента **Языковая модель** к компоненту **Вывод чата**.

8. В конце потока добавьте ещё один компонент **История сообщений** и установите его в режим **Store**.

При необходимости настройте дополнительные параметры этого компонента с учётом того, что он будет сохранять сообщения, а не извлекать их.

9. Подключите выход компонента **Вывод чата** к входу **Сообщения** компонента **История сообщений**.

Каждый ответ языковой модели проходит из компонента **Языковая модель** в компонент **Вывод чата**, а затем сохраняется в памяти чата последним компонентом **История сообщений**.

Чтобы сохранять и извлекать память чата из выделенной внешней базы данных, используйте компонент **История сообщений** совместно с компонентом памяти чата, специфичным для провайдера.

Следующие шаги описывают создание потока, использующего компонент **Память чата Redis**. Другие варианты: компонент **Mem0 Память чата** и компонент **Cassandra Память чата**.

1. Создайте или откройте поток, в котором хотите использовать память чата.
2. В начале потока добавьте компонент **История сообщений** и компонент **Память чата Redis**:

1. Настройте компонент **Память чата Redis** для подключения к вашей базе данных Redis. Для получения дополнительной информации см. документацию Redis.
2. Установите компонент **История сообщений** в режим **Retrieve**.
3. В панели инспекции компонента **История сообщений** включите **Внешняя память**. В разделе **Controls** также можно настроить параметры сортировки, фильтрации и ограничения памяти.
4. Подключите выход компонента **Память чата Redis** к входу **Внешняя память** компонента **История сообщений**.

3. Добавьте компонент **Шаблон промпта**, укажите переменную {memory} в поле **Шаблон**, затем подключите выход **История сообщений** к входу **memory**.

Компонент **Шаблон промпта** передаёт инструкции и контекст в языковую модель отдельно от сообщений чата, поступающих через компонент **Ввод в чат**. Шаблон может содержать любой текст и переменные, например:

You are a helpful assistant that answers questions.

Use markdown to format your answer, properly embedding images and urls.

History:

{memory}

Переменные в формате {variable} динамически создают поля в компоненте **Шаблон промпта**, через которые поток получает значения из других компонентов, глобальных переменных Генеративный ИИ или ввода во время выполнения. Для получения дополнительной информации см. [Определение переменных в промптах](#).

В этом примере переменная {memory} заполняется извлечёнными данными памяти чата, которые затем передаются в компонент **Языковая модель** или **Агент** как дополнительный контекст.

4. Подключите выход компонента **Шаблон промпта** к входу **Системное сообщение** компонента **Языковая модель**.

В этом примере центральным драйвером чата служит компонент **Языковая модель**, но можно использовать другой компонент языковой модели или компонент **Агент**.

5. Добавьте компонент **Ввод в чат** и подключите его к входу **Ввод** компонента **Языковая модель**.
6. Подключите выход компонента **Языковая модель** к компоненту **Вывод чата**.
7. В конце потока добавьте ещё одну пару компонентов **История сообщений** и **Память чата Redis**:

1. Настройте компонент **Память чата Redis** для подключения к вашей базе данных Redis.
2. Установите компонент **История сообщений** в режим **Store**.
3. В панели инспекции компонента **История сообщений** включите **Внешняя память**. При необходимости настройте дополнительные параметры с учётом того, что данный компонент будет сохранять сообщения, а не извлекать их.
4. Подключите выход компонента **Память чата Redis** к входу **Внешняя память** компонента **История сообщений**.

8. Подключите выход компонента **Вывод чата** к входу **Сообщения** компонента **История сообщений**.

Каждый ответ языковой модели проходит из компонента **Языковая модель** в компонент **Вывод чата**, а затем сохраняется в памяти чата через компоненты **История сообщений** и **Память чата Redis**.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Доступные параметры зависят от режима компонента — **Получение** или **Сохранение**.

Режим получения

Название	Тип	Описание	
template	String	Входной параметр. Шаблон для форматирования данных. Может содержать ключи {text}, {sender} или любой другой ключ из данных сообщения.	

Название	Тип	Описание
memory	Внешняя память	Входной параметр. Источник внешней памяти для получения сообщений. Если не задан, используется хранилище Генеративный ИИ.
n_messages	Integer	Входной параметр. Количество сообщений для получения. По умолчанию: 100.
order	String	Входной параметр. Порядок сортировки сообщений. По умолчанию: Ascending.
sender_type	String	Входной параметр. Фильтрация по типу отправителя: User, Machine или Machine and User (по умолчанию).
session_id	String	Входной параметр. ID сеанса памяти чата для получения. Если не задан, используется текущий ID сеанса выполнения потока.
Режим сохранения		
Название	Тип	Описание
template	String	Входной параметр. Шаблон для форматирования данных. Может содержать ключи {text}, {sender} или любой другой ключ из данных сообщения.
message	String	Входной параметр. Сообщение для сохранения. Обычно поступает при подключении компонента
memory	Внешняя память	Вывод чата. Входной параметр. Цель внешней памяти для сохранения сообщений. Если не задана, используется хранилище Генеративный ИИ.

Название	Тип	Описание
sender	String	Входной параметр. Фильтрация сохраняемых сообщений по отправителю: User, Machine или Machine and User (по умолчанию).
sender_name	String	Входной параметр. Резервная метка отправителя, используемая, если у сообщения отсутствуют метаданные отправителя.
session_id	String	Входной параметр. ID сеанса памяти чата для сохранения. Если не задан, используется текущий ID сеанса выполнения потока. Используйте пользовательские ID сеансов для разделения памяти чата между разными пользователями или приложениями, запускающими один и тот же поток.
sender_type	String	Входной параметр. Фильтрация по типу отправителя: User, Machine или Machine and User (по умолчанию).

Вывод компонента

Память можно получить в одном из двух форматов:

- **Сообщения** — возвращает память в виде объектов Message, включая messages_text с текстом сообщений чата. Это стандартный формат для передачи памяти как сообщений чата другому компоненту.
- **DataFrame** — возвращает память в виде DataFrame с данными сообщений. Полезно, когда память нужно получить в табличном формате, а не в виде сообщений чата.

Тип вывода можно выбрать рядом с выходным портом компонента.

25.8 Утилиты

25.8.1 Калькулятор

Компонент **Калькулятор** выполняет базовые арифметические операции над математическими выражениями. Он поддерживает сложение, вычитание, умножение, деление и возведение в степень.

Пример использования этого компонента в потоке см. в разделе **Компонент Интерпретатор Python**.

Параметры компонента «Калькулятор»

Название	Тип	Описание
expression	String	Входной параметр. Арифметическое выражение для вычисления, например $4*4*(33/22)+12-20$.
result	Data	Выходной параметр. Результат вычисления в виде объекта Data, содержащего вычисленное значение.

25.8.2 Текущая дата

Компонент **Текущая дата** возвращает текущие дату и время в указанном часовом поясе. Компонент предоставляет гибкий способ получения информации о дате и времени с учётом часового пояса в рамках конвейера Генеративный ИИ.

Параметры компонента

Название	Тип	Описание
timezone	Строка	Входной параметр. Часовой пояс для текущей даты и времени.
current_date	Строка	Выходной параметр. Полученные текущие дата и время в выбранном часовом поясе.

25.8.3 Интерпретатор Python

Компонент **Интерпретатор Python** позволяет выполнять код Python с импортированными пакетами.

Компонент может импортировать только те пакеты, которые уже установлены в вашей среде Генеративный ИИ. Если при попытке использовать пакет возникает ImportError, необходимо сначала его установить.

Чтобы установить пользовательские пакеты, см. **Установка пользовательских зависимостей**.

Использование компонента в потоке

1. В поле **Глобальный импорт** добавьте пакеты, которые хотите импортировать, в виде списка, разделённого запятыми, например math,pandas. Требуется как минимум один импорт.
2. В поле **Код Python** введите код Python для выполнения. Используйте print(), чтобы увидеть вывод.
3. Необязательно: включите **Режим инструмента** и подключите компонент **Интерпретатор Python** к компоненту **Агент** в качестве инструмента. Например, подключите **Интерпретатор Python** и компонент **Калькулятор** как инструменты к компоненту **Агент** и проверьте, как агент выбирает разные инструменты для решения математических задач.

4. Задайте агенту простой математический вопрос. Компонент **Калькулятор** может выполнять сложение, вычитание, умножение, деление и возведение в степень. Агент использует инструмент `evaluate_expression` для ответа на вопрос.

Результат:

Executed evaluate_expression

Input:

```
{
  "expression": "2+5"
}
```

Output:

```
{
  "result": "7"
}
```

5. Передайте агенту полный код Python. В следующем примере создаётся таблица Pandas DataFrame с импортированными пакетами pandas и вычисляется квадратный корень из среднего значения квадратов:

```
import pandas as pd
import math
```

```
# Create a simple DataFrame
df = pd.DataFrame({
  'numbers': [1][2][3][4][5]
  'squares': [x**2 for x in range(1, 6)]
})
```

```
# Calculate the square root of the mean
result = math.sqrt(df['squares'].mean())
print(f"Square root of mean squares: {result}")
```

Агент выбирает инструмент `run_python_repl` для решения задачи.

Результат:

Executed run_python_repl

Input:

```
{
  "python_code": "import pandas as pd\nimport math\n\n# Create a simple DataFrame\ndf = pd.DataFrame({\n  'numbers':,\n  'squares': [x**2 for x in range(1, 6)]\n})\n\n# Calculate the square root of the mean\nresult =\nmath.sqrt(df['squares'].mean())\nprint(f\"Square root of mean squares: {result}\")"
}
```

Output:

```
{
  "result": "Square root of mean squares: 3.3166247903554"
}
```

Если вы не укажете импорт пакетов в чате, агент всё равно сможет создать таблицу с `pd.DataFrame`, поскольку пакет pandas импортируется глобально компонентом **Интерпретатор Python** через поле **Глобальный импорт**.

Передача входных данных в компонент

Чтобы передать входные данные в компонент **Интерпретатор Python**, необходимо изменить код компонента, добавив поля ввода. После добавления поля ввода в код компонента появляется соответствующий порт для соединений.

Например, чтобы подключить компонент **Ввод текста** и передать значение URL в компонент **Интерпретатор Python**:

1. Добавьте компонент **Интерпретатор Python** в ваш поток.
2. Нажмите **Редактировать код**, чтобы открыть редактор кода компонента.
3. Внесите следующие изменения:

- Добавьте поле ввода URL в список inputs. Это создаёт входной порт, к которому могут подключаться другие компоненты.
- Обновите метод get_globals, чтобы извлечь значение URL и добавить его в словарь globals. Это делает переменную url доступной в Python-коде компонента.
- Обновите значение кода Python по умолчанию, чтобы использовать переменную url.

Пример кода:

```
import importlib
```

```
from langchain_experimental.utilities import PythonREPL
from lfx.custom.custom_component.component import Component
from lfx.io import MultilineInput, Output, StrInput
from lfx.schema.data import Data
from lfx.schema.message import Message # Needed to extract text from Message objects
```

```
class PythonREPLComponent(Component):
```

```
    display_name = "Python Interpreter"
```

```
    description = "Run Python code with optional imports. Use print() to see the output."
```

```
    documentation: str = "https://docs.langflow.org/python-interpreter"
```

```
    icon = "square-terminal"
```

```
    inputs = [
```

```
        StrInput(
```

```
            name="global_imports",
```

```
            display_name="Global Imports",
```

```
            info="A comma-separated list of modules to import globally, e.g. 'math,numpy,pandas'.",
```

```
            value="math,pandas",
```

```
            required=True,
```

```
        ),
```

```
        MultilineInput(
```

```
            name="python_code",
```

```
            display_name="Python Code",
```

```
            info="The Python code to execute. Only modules specified in Global Imports can be used. Use 'url' variable if URL input is connected.",
```

```
            value="print(f'URL: {url}')" # Updated to make the URL variable available to the Python code execution
```

```
            input_types=["Message"],
```

```
            tool_mode=True,
```

```
            required=True,
```

```
        ),
```

```
        # Add the URL input field to inputs list
```

```
        StrInput(
```

```
            name="url",
```

```
            display_name="URL",
```

```
            info="URL variable that can be used in Python code. Connect a Text component or enter manually.",
```

```
            value="",
```

```
            input_types=["Text", "Message"],
```

```
            required=False,
```

```
        ),
```

```
    ]
```

```
    outputs = [
```

```
        Output(
```

```
            display_name="Results",
```

```
            name="results",
```

```
            type_=Data,
```

```
            method="run_python_repl",
```

```
        ),
```

```
    ]
```



```

def get_globals(self, global_imports: str | list[str]) -> dict:
    """Create a globals dictionary with only the specified allowed imports and input variables."""
    global_dict = {}

    try:
        if isinstance(global_imports, str):
            modules = [module.strip() for module in global_imports.split(",")]
        elif isinstance(global_imports, list):
            modules = global_imports
        else:
            msg = "global_imports must be either a string or a list"
            raise TypeError(msg)

        for module in modules:
            try:
                imported_module = importlib.import_module(module)
                global_dict[imported_module.__name__] = imported_module
            except ImportError as e:
                msg = f"Could not import module {module}: {e!s}"
                raise ImportError(msg) from e

        # Add the URL variable to the component's globals dictionary
        # Extract from Message object or use the string directly
        if hasattr(self, "url") and self.url:
            url_value = self.url.text if isinstance(self.url, Message) else str(self.url)
            if url_value:
                global_dict["url"] = url_value # Makes 'url' available in Python code
                self.log(f"URL variable set: {url_value}")

    except Exception as e:
        self.log(f"Error in global imports: {e!s}")
        raise
    else:
        self.log(f"Successfully imported modules: {list(global_dict.keys())}")
        return global_dict

def run_python_repl(self) -> Data:
    try:
        # Extract Python code text if it's a Message object
        python_code_text = self.python_code
        if isinstance(python_code_text, Message):
            python_code_text = python_code_text.text if python_code_text.text else ""
        elif not isinstance(python_code_text, str):
            python_code_text = str(python_code_text)

        globals_ = self.get_globals(self.global_imports)
        python_repl = PythonREPL(_globals=globals_)
        result = python_repl.run(python_code_text)
        result = result.strip() if result else ""

        self.log("Code execution completed successfully")
        return Data(data={"result": result})

    except ImportError as e:
        error_message = f"Import Error: {e!s}"
        self.log(error_message)
        return Data(data={"error": error_message})

    except SyntaxError as e:
        error_message = f"Syntax Error: {e!s}"
        self.log(error_message)
        return Data(data={"error": error_message})

```

```
except (NameError, TypeError, ValueError) as e:
    error_message = f"Error during execution: {e!s}"
    self.log(error_message)
    return Data(data={"error": error_message})
```

```
def build(self):
    return self.run_python_repl
```

4. Нажмите **Проверить и сохранить**, чтобы применить изменения.
5. Добавьте компонент **Ввод текста** в ваш поток и задайте его значение, например google.com.
6. Подключите выход компонента **Ввод текста** к полю ввода **URL** в вашем компоненте **Интерпретатор Python**.

Теперь компонент **Интерпретатор Python** может использовать переменную url в выполняемом Python-коде.

Параметры компонента

Название	Тип	Описание
global_imports	String	Входной параметр. Список модулей, разделённых запятыми, которые импортируются глобально, например math,pandas,numpy.
python_code	Код	Входной параметр. Python-код для выполнения. Можно использовать только модули, указанные в поле Глобальный импорт .
results	Data	Выходной параметр. Вывод выполненного Python-кода, включая напечатанные результаты или ошибки.

25.8.4 База данных SQL

Компонент **База данных SQL** выполняет SQL-запросы к SQLAlchemy-совместимым базам данных. Он поддерживает любые совместимые с SQLAlchemy СУБД, такие как PostgreSQL, MySQL и SQLite.

Для запросов CQL см. пакет **DataStax**.

Запрос к базе данных SQL с запросами на естественном языке

В следующем примере показано, как использовать компонент **База данных SQL** в потоке, а затем расширить его, чтобы поддерживать запросы на естественном языке через компонент **Агент**.

Это позволяет использовать один и тот же компонент **База данных SQL** для любого запроса, не ограничиваясь одним вручную введённым запросом и не требуя от пользователя, приложения или другого компонента предоставления корректного SQL-синтаксиса. Пользователям не нужно знать SQL, поскольку компонент **Агент** переводит запросы на естественном языке в SQL-запросы, передаёт их компоненту **База данных SQL** и возвращает результаты пользователю.

Кроме того, входные данные от приложений и других компонентов не нужно извлекать и преобразовывать в точные SQL-запросы: достаточно предоставить агенту достаточно контекста, чтобы он понял, что необходимо создать и выполнить SQL-запрос.

1. Используйте собственную базу данных или создайте тестовую.

Создание тестовой базы данных SQL

1. Создайте базу данных с именем test.db:

```
sqlite3 test.db
```

2. Добавьте несколько записей в базу данных:

```
sqlite3 test.db "
```

```
CREATE TABLE users (  
  id INTEGER PRIMARY KEY,  
  name TEXT,  
  email TEXT,  
  age INTEGER  
);
```

```
INSERT INTO users (name, email, age) VALUES
```

```
('John Doe', 'john@example.com', 30),  
(Jane Smith', 'jane@example.com', 25),  
(Bob Johnson', 'bob@example.com', 35);
```

```
"
```

3. Проверьте, что база данных создана и содержит ваши данные:

```
sqlite3 test.db "SELECT * FROM users;"
```

Результат должен отобразить данные, введенные на предыдущем шаге:

```
1|John Doe|john@example.com|30  
2|Jane Smith|jane@example.com|25  
3|Bob Johnson|bob@example.com|35
```

2. Добавьте компонент **База данных SQL** в ваш поток.

3. В поле **URL-адрес базы данных** укажите строку подключения к вашей базе данных, например sqlite:///test.db.

На этом этапе вы можете ввести SQL-запрос в поле **SQL-запрос** или использовать порт, чтобы передать запрос из другого компонента, например **Ввод в чат**. Если нужно больше места, нажмите **Expand**, чтобы открыть поле ввода в полноэкранном режиме. Чтобы сделать компонент более динамичным в агентном контексте, используйте компонент **Агент** для преобразования ввода на естественном языке в SQL-запросы, как описано в следующих шагах.

4. Нажмите на компонент **База данных SQL**, чтобы открыть меню заголовка компонента, а затем включите **Режим инструмента**.

Теперь этот компонент можно использовать как инструмент для агента. В **Режиме инструмента** запрос не задаётся в компоненте **База данных SQL** напрямую: агент сам сгенерирует и отправит запрос, если определит, что инструмент необходим для ответа на запрос пользователя. Для получения дополнительной информации см. Настройка инструментов для агентов.

5. Добавьте компонент **Агент** в ваш поток и укажите ваш API-ключ OpenAI.

По умолчанию используется модель OpenAI. Если вы хотите использовать другую модель, измените поля **Поставщик модели**, **Название модели** и **Ключ API** соответственно.

Если необходимо выполнять высокоспециализированные запросы, рассмотрите выбор модели, обученной на соответствующих задачах, например на сложных SQL-запросах. Если нужная модель отсутствует в встроенном списке компонента **Агент**, установите

Поставщик модели в значение **Connect other models** и подключите любой компонент языковой модели.

- 6. Подключите выход **Toolset** компонента **База данных SQL** ко входу **Инструменты** компонента **Агент**.
- 7. Нажмите **Playground** и задайте агенту вопрос о данных в вашей базе данных, например Which users are in my database?.

Агент определяет, что для ответа на вопрос необходимо выполнить запрос к базе данных, использует LLM для генерации SQL-запроса, а затем выполняет его через действие RUN_SQL_QUERY компонента **База данных SQL** и возвращает результаты в разговорном формате — если только вы не укажете возвращать необработанные данные или использовать другой формат.

В следующем примере был выполнен запрос к тестовой базе данных с небольшим объёмом данных, однако при более полном наборе данных можно задавать более детальные или сложные вопросы.

Here are the users in your database:

- 1. **John Doe** - Email: john@example.com
- 2. **Jane Smith** - Email: jane@example.com
- 3. **Bob Johnson** - Email: bob@example.com

It seems there are duplicate entries for the users.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
database_url	URL-адрес базы данных	Входной параметр. URL подключения к базе данных, совместимый с SQLAlchemy.
query	SQL-запрос	Входной параметр. SQL-запрос для выполнения: может быть введён напрямую, передан из другого компонента или, в Режиме инструмента , автоматически предоставлен компонентом Агент .
include_columns	Включить столбцы	Входной параметр. Определяет, включать ли имена столбцов в результат. По умолчанию включено (true).
add_error	Добавить ошибку	Входной параметр. Если включено, добавляет сообщения об ошибках в

Название	Отображаемое название	Описание
run_sql_query	Результирующая таблица	результат, если они возникают. По умолчанию отключено (false). Выходной параметр. Результаты запроса в виде DataFrame.

25.8.5 Устаревшие основные компоненты

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, но рекомендуется заменить их поддерживаемыми компонентами как можно скорее. Рекомендуемые замены указаны в баннере **Устаревшее** на компонентах в ваших потоках. Они также указаны в примечаниях к выпуску и документации Генеративный ИИ, когда это возможно.

Если вы не уверены, как заменить устаревший компонент, **Поиск** для компонентов по поставщику, сервису или названию компонента. Компонент мог быть объявлен устаревшим в пользу полностью нового компонента, похожего компонента или новой версии того же компонента в другой категории.

Если очевидной замены нет, рассмотрите возможность адаптации другого компонента под ваш сценарий. Например, многие **Основные компоненты** предоставляют общую функциональность, поддерживающую несколько поставщиков и сценариев использования, такие как компонент **API-запрос**.

Если ни один из вариантов не подходит, вы можете использовать код устаревшего компонента для создания собственного пользовательского компонента или начать обсуждение об этом компоненте.

Чтобы не поощрять использование устаревших компонентов в новых потоках, эти компоненты по умолчанию скрыты. В визуальном редакторе вы можете нажать **Основные компоненты**, чтобы переключить фильтр **Устаревшее**.

Устаревшие Данные компоненты

Следующие Данные компоненты находятся в статусе устаревших:

- **Загрузить CSV**
- **Загрузить JSON**

Замените эти компоненты на Компонент **Чтение файла**, который поддерживает загрузку CSV и JSON файлов, а также многих других типов файлов.

Устаревшие вспомогательные компоненты

Следующие вспомогательные компоненты находятся в статусе устаревших:

- **Хранилище сообщений**: Заменён на Компонент **История сообщений**
- **Создать список**: Заменить на Компоненты обработки
- **Генератор идентификаторов**: Заменить на компонент, который выполняет произвольный код для генерации ID или встраивает скрипт генератора ID в ваш код приложения (вне потоков Генеративный ИИ).
- **Выходной анализатор**: Заменить на Компонент **Структурированный вывод** и Компонент **Парсер**. Необходимые компоненты зависят от типов данных и сложности задачи парсинга.

Компонент **Выходной анализатор** преобразовывал вывод языковой модели в формат значений, разделённых запятыми (CSV), например ["item1", "item2", "item3"], используя CommaSeparatedListOutputParser из LangChain. Компонент **Структурированный вывод** является хорошей альтернативой, поскольку он также форматирует ответы LLM с поддержкой пользовательских схем и более сложного парсинга.

Компоненты **Парсинг** предоставляют только инструкции по форматированию и функции парсинга. *Они не включают подсказки.* Вы должны подключить парсеры к **Шаблон промпта** компонентам, чтобы создать подсказки, которые могут использовать LLM.

Устаревшие логические компоненты

Следующие логические компоненты находятся в статусе устаревших:

Результат

В качестве альтернативы этому устаревшему компоненту смотрите **Если-иначе** компонент.

Компонент **Условный роутер** маршрутизирует объекты Data на основе условия, применяемого к указанному ключу, включая проверку булевых значений. Он поддерживает true_output и false_output для маршрутизации результатов в зависимости от оценки условия.

Этот компонент полезен в рабочих процессах, требующих условной маршрутизации сложных структур данных, позволяя принимать динамические решения на основе содержимого данных.

Он может обрабатывать как один объект Data, так и список объектов Data. При обработке списка объектов Data выполняются следующие действия:

- Каждый объект в списке оценивается отдельно.
- Объекты, удовлетворяющие условию, направляются в true_output.
- Объекты, не удовлетворяющие условию, направляются в false_output.
- Если все объекты направляются в один вывод, другой вывод будет пустым.

Компонент **Условный роутер** принимает следующие параметры:

Название	Type	Описание
data_input	Данные	Параметр Ввод. Объект Данные или список объектов Данные для обработки. Этот ввод может обрабатывать как отдельные элементы, так и списки.
key_name	String	Параметр Ввод. Имя ключа в объекте Данные, который нужно проверить.
operator	Dropdown	Параметр Ввод. Оператор для применения. Варианты: equals, not equals, contains, starts with, ends with, boolean validator. По умолчанию: equals.

Название	Type	Описание
compare_value	String	Параметр Ввод. Значение для сравнения. Не отображается/не используется, когда оператор boolean validator.

Опции operator имеют следующие поведения:

- equals: Точное сравнение значения ключа с compare_value.
- not equals: Обратное точному совпадению.
- contains: Проверка, содержится ли compare_value в значении ключа.
- starts with: Проверка, начинается ли значение ключа с compare_value.
- ends with: Проверка, заканчивается ли значение ключа на compare_value.
- boolean validator: Рассматривает значение ключа как булево. Следующие значения считаются истинными:
 - Булево true.
 - Строки: true, 1, yes, y, on (без учёта регистра)
 - Любое другое значение преобразуется с помощью функции bool() из Python

Пропуск

В качестве альтернативы этому устаревшему компоненту используйте **Если-иначе** компонент для передачи сообщения без изменения.

Компонент **Пропуск** пересылает входное сообщение без изменения.

Он принимает следующие параметры:

Название	Display Название	Info
input_message	Входное сообщение	Входной параметр. Сообщение для пересылки.
ignored_message	Проигнорированное сообщение	Входной параметр. Второе сообщение, которое игнорируется. Используется как обходной путь для непрерывности.
output_message	Вывод Сообщения	Выходной параметр. Пересланное сообщение из входа.

Поток как Инструмент

Этот компонент создал инструмент из функции, которая запускала загруженный поток. Он был устаревшим в версии Генеративный ИИ 1.1.2 и заменён Компонент **Запустить поток**.

Подпоток

Этот компонент интегрировал целые потоки как компоненты в более крупный рабочий процесс. Он динамически генерировал входные данные на основе выбранного потока и выполнял поток с предоставленными параметрами.

Он был устаревшим в версии Генеративный ИИ 1.1.2 и заменён Компонентом **Запустить поток**.

Устаревшие компоненты обработки

Следующие компоненты обработки находятся в устаревшем статусе:

Изменение метаданных

Замените этот устаревший компонент на **Операции с данными** компонент.

Этот компонент изменяет метаданные входных объектов. Он может добавлять новые метаданные, обновлять существующие метаданные и удалять указанные поля метаданных. Компонент работает как с объектами Message, так и с объектами Data и также может создать новый объект Data из предоставленного пользователем текста.

Он принимает следующие параметры:

Название	Отображать Название	Информация
input_value	Ввод	Входной параметр. Объекты, к которым следует добавить Метаданные.
Текст_in	Пользовательский текст	Входной параметр. Текст ввод; значение содержится в атрибуте 'Текст' объекта Data. Пустые записи Текст игнорируются.
metadata	Метаданные	Входной параметр. Метаданные для добавления к каждому объекту.
remove_fields	Поля для удаления	Входной параметр. Поля Метаданные для удаления.
data	Данные	Выходной параметр. Список объектов Ввод, каждый с добавленными метаданными.

Объединить данные

Замените этот устаревший компонент на **Операции с данными** компонент или на **Цикл** компонент.

Этот компонент объединяет несколько источников данных в единый объект Data.

Компонент проходит по списку объектов Data, объединяя их в один объект Data (merged_data). Если список входных данных пуст, он возвращает пустой объект данных. Если входных объектов данных только один, он возвращает этот объект без изменений.

Процесс объединения использует оператор сложения для комбинирования объектов данных.

Объединить текст

Замените этот устаревший компонент на **Операции с данными** компонент.

Этот компонент конкатенирует два входных Текст в один блок Текст с использованием указанного разделителя, выводя объект Message с объединённым текстом.

Создать данные

Замените этот устаревший компонент на **Динамическое создание данных** компонент. Этот компонент динамически создает объект Data с указанным количеством полей и ключом Текст.

Он принимает следующие параметры:

Название	Отображать Название	Информация
number_of_fields	Количество полей	Параметр Ввод. Количество полей, которые будут добавлены в запись.
Текст_key	Текстовый ключ	Параметр Ввод. Ключ, который определяет поле, используемое в качестве содержимого Текст.
Текст_key_validator	Валидатор текстового ключа	Параметр Ввод. Если включено, проверяет, присутствует ли указанный Text Key в данном Data.

Данные в DataFrame/Данные в Сообщения

Замените эти устаревшие компоненты на более новые компоненты обработки, такие как **Операции с данными** компонент и **Конвертировать Тип** компонент.

Эти компоненты преобразовывали один или несколько объектов Data в объект DataFrame или Message.

Для компонента **Данные to DataFrame** каждый объект Data соответствует одной строке в результирующем DataFrame. Поля из атрибута .data становятся столбцами, а поле .text (если присутствует) помещается в столбец text.

Извлечь ключ

Замените этот устаревший компонент на компонент **Операции с данными**.

Этот компонент извлекает определённый ключ из объекта Data и возвращает значение, связанное с этим ключом.

Фильтрация данных

Замените этот устаревший компонент на компонент **Операции с данными**.

Этот компонент фильтрует объект Data на основе списка ключей (filter_criteria), возвращая новый объект Data (filtered_data), который содержит только пары ключ-значение, соответствующие критериям фильтра.

Значения фильтра

Замените этот устаревший компонент на компонент **Операции с данными**.

Компонент Filter values фильтрует список элементов данных на основе указанного ключа, значения фильтра и оператора сравнения.

Он принимает следующие параметры:

Название	Отображать Название	Информация
input_data	Ввод данные	Входной параметр Список элементов данных для фильтрации.
filter_key	Ключ фильтра	Входной параметр Ключ, по которому производится фильтрация.
filter_value	Значение фильтра	Входной параметр Значение, по которому производится фильтрация.

Название	Отображать Название	Информация
operator	Оператор сравнения	Входной параметр Оператор, применяемый для сравнения значений.
filtered_data	Отфильтрованные данные	Выходной параметр. Полученный список отфильтрованных элементов данных.

Очиститель JSON

Замените этот устаревший компонент на компонент **Парсер**.

Этот компонент очищает строки JSON, чтобы гарантировать их полное соответствие спецификации JSON.

Он принимает следующие параметры:

Название	Отображать Название	Информация
json_str	JSON-строка	Входной параметр Строка JSON, которую необходимо очистить. Это может быть необработанная, потенциально некорректная строка JSON, полученная от языковых моделей или других источников, которые могут не полностью соответствовать спецификации JSON.
remove_control_chars	Удалить управляющие символы	Входной параметр Если установлено в True, эта опция удаляет управляющие символы (символы ASCII 0-31 и 127) из строки JSON. Это может помочь избавиться от невидимых символов, которые могут вызвать проблемы при разборе или сделать JSON недействительным.
normalize_unicode	Нормализовать Unicode	Входной параметр При включении эта опция нормализует Unicode-символы в строке JSON до их канонической формы композиции (NFC). Это обеспечивает

Название	Отображать Название	Информация
validate_json	Проверить JSON	согласованное представление Unicode-символов в разных системах и предотвращает потенциальные проблемы с кодировкой символов. Входной параметр Если установлено в True, эта опция пытается разобрать строку JSON, чтобы убедиться в её корректности, перед выполнением окончательной операции исправления. При недействительном JSON генерируется ValueError, позволяя раннее обнаружение серьезных структурных проблем в JSON. Выходной параметр. Полученная очищенная, исправленная и проверенная строка JSON, полностью соответствующая спецификации JSON.
output	Очищенный JSON-строка	

Преобразование типа Сообщения в Данные

Замените этот устаревший компонент на компонент **Конвертировать Тип**.

Этот компонент преобразует объекты Message в объекты Data.

DataFrame парсер

Замените этот устаревший компонент на **Операции с DataFrame** компонент или **Парсер** компонент.

Этот компонент преобразует объекты DataFrame в обычный Текст с использованием шаблонов.

Он принимает следующие параметры:

Название	Отображать Название	Информация
df	DataFrame	Входной параметр DataFrame, который необходимо преобразовать в строки Текст.
template	Шаблон	Входной параметр Шаблон для

Название	Отображать Название	Информация форматирования (используйте заполнители {column_name}).
sep	Разделитель	Входной параметр Строка для объединения строк в выводе.
Текст	Текст	Выходной параметр. Все строки объединены в единый текст.

JSON парсер

Замените этот устаревший компонент на **Парсер** компонент.

Этот компонент преобразует и извлекает поля JSON в объектах Message и Data с помощью запросов JQ, затем возвращает filtered_data, который представляет собой список объектов Data.

Экстрактор регулярных выражений

Замените этот устаревший компонент на **Парсер** компонент.

Этот компонент извлекает шаблоны в Текст с помощью регулярных выражений. Он может использоваться для поиска и извлечения конкретных шаблонов или информации в тексте.

Выбрать данные

Замените этот устаревший компонент на **Операции с данными** компонент.

Этот компонент выбирает один объект Data из списка.

Он принимает следующие параметры:

Название	Отображать Название	Информация
data_list	Список данных	Входной параметр. Список данных для выбора
data_index	Индекс данных	Входной параметр. Индекс данных для выбора
selected_data	Выбранный Данные	Выходной параметр. Выбранный объект Data

Результат

Замените этот устаревший компонент на **Операции с данными** компонент.

Этот компонент динамически обновляет или добавляет данные с указанными полями.

Он принимает следующие параметры:

Название	Отображать Название	Информация
old_data	Данные	Входной параметр. Записи для обновления
number_of_fields	Количество полей	Входной параметр. Количество полей для добавления. Максимум — 15
Текст_key	Текстовый ключ	Входной параметр. Ключ для содержимого Текст

Название	Отображать Название	Информация
Текст_key_validator	Валидатор текстового ключа	Входной параметр. Проверка наличия ключа Текст
data	Данные	Выходной параметр. Обновленные объекты Данные

Устаревшие Инструменты компоненты

Следующие компоненты инструментария находятся в устаревшем состоянии: - **Калькулятор Инструмент**: Заменён на Компонент **Калькулятор**. - **Структурированный код Python**: Заменён на Компонент **Интерпретатор Python**. - **Python REPL**: Заменён на Компонент **Интерпретатор Python**. - **Поисковый API**: Заменён на Компонент **SearchApi**. - **SearXNG Поиск**: Нет прямой замены. Используйте поисковый компонент другого провайдера, создайте пользовательский компонент или используйте основной компонент, например **API-запрос** компонент. - **Serp поисковый API**: Заменить на **SerpApi** пакет. - **Tavily поисковый API**: Заменён на **Tavily** пакет. - **API Викиданных**: Заменён на **Wikipedia** пакет. - **Wikipedia API**: Заменён на **Wikipedia** пакет. - **Yahoo! Финансы**: Заменён на пакет **Yahoo! Поиск**.

26. Бандлы (интеграции)

26.1 О пакетах

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ». Добавляйте их в свои потоки и настраивайте так же, как основные компоненты Генеративный ИИ.

Чтобы просмотреть пакеты, нажмите **Bundles** в визуальном редакторе.

Поддержка и документация пакетов

Многие компоненты из пакетов разрабатываются сторонними участниками проекта Генеративный ИИ.

Некоторые поставщики предоставляют документацию вместе со своими пакетами, другие документируют их в собственной документации. У ряда пакетов документация отсутствует.

Чтобы найти документацию для конкретного компонента из пакета, просмотрите документацию Генеративный ИИ и документацию поставщика. При наличии ссылки на соответствующую документацию, например на конечные точки API, можно найти непосредственно в самом компоненте:

1. Щёлкните по компоненту, чтобы открыть панель инспекции компонента.
2. Нажмите **Больше**.
3. Выберите **Документы**.

Документация Генеративный ИИ сосредоточена на использовании пакетов внутри потоков: описывает шаги конфигурации, специфичные для модуля «Генеративный ИИ». Для получения сведений о функциях или API поставщика обращайтесь к документации поставщика.

Параметры компонента

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Основные компоненты и пакеты

Документация Генеративный ИИ не содержит исчерпывающего списка пакетов и компонентов в них. Для получения актуального списка пакетов и компонентов вашей версии Генеративный ИИ проверьте раздел **Bundles** в визуальном редакторе.

Если вы не можете найти компонент, который использовали в более ранней версии Генеративный ИИ, он мог быть удалён или помечен как устаревший. Генеративный ИИ предлагает общие **Основные компоненты** в дополнение к сторонним пакетам, специфичным для поставщика.

Чтобы найти конкретный сервис или интеграцию, воспользуйтесь **поиском** в визуальном редакторе.

Если ничего не помогает, вы всегда можете создать собственные пользовательские компоненты.

Устаревшие пакеты

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, однако рекомендуется как можно скорее заменить их поддерживаемыми компонентами.

Рекомендуемые замены указаны в баннере **Устаревшее** на компонентах в ваших потоках, а также в примечаниях к выпуску и документации Генеративный ИИ там, где это возможно.

Если вы не уверены, как заменить устаревший компонент, воспользуйтесь **поиском** по имени поставщика, сервиса или компонента. Компонент мог быть объявлен устаревшим в пользу нового компонента, похожего компонента или новой версии того же компонента в другой категории.

Если очевидной замены нет, рассмотрите возможность адаптации другого компонента под ваш сценарий. Например, многие **Основные компоненты** предоставляют общую функциональность, поддерживающую нескольких поставщиков и сценарии использования, — в частности, компонент **API-запрос**.

Если ни один из вариантов не подходит, вы можете использовать код устаревшего компонента для создания собственного пользовательского компонента или начать обсуждение об этом устаревшем компоненте.

Чтобы не поощрять использование устаревших компонентов в новых потоках, они скрыты по умолчанию. В визуальном редакторе можно нажать **Настройки компонента** и переключить фильтр **Устаревшее**.

Следующие пакеты содержат только устаревшие компоненты.

Пакет CrewAI

Замените устаревшие компоненты CrewAI другими агентными компонентами, например компонентом **Агент**.

Агент CrewAI

Этот компонент представляет агентов CrewAI и позволяет создавать специализированных AI-агентов с определёнными ролями, целями и возможностями внутри команды. Для получения дополнительной информации см. документацию по агентам CrewAI.

Компонент принимает следующие параметры:

Название	Отображаемое название	Описание	
role	Роль	Входной	параметр.
		Роль агента.	
goal	Цель	Входной	параметр.
		Цель агента.	
backstory	Предыстория	Входной	параметр.
		Предыстория агента.	
tools	Инструменты	Входной	параметр.
		Инструменты, доступные агенту.	
llm	Языковая модель	Входной	параметр.
		Языковая модель, управляющая агентом.	
memory	Память	Входной	параметр.
		Определяет, будет ли у агента память.	
verbose	Подробный вывод	Входной	параметр.
		Включает подробный вывод.	

Название	Отображаемое название	Описание
allow_delegation	Разрешить делегирование	Входной параметр. Определяет, может ли агент делегировать задачи другим агентам.
allow_code_execution	Разрешить выполнение кода	Входной параметр. Определяет, может ли агент выполнять код.
agent_kwargs	Именованные аргументы	Входной параметр. Дополнительные именованные аргументы для агента.
output	Агент	Выходной параметр. Сконструированный объект агента CrewAI.

CrewAI Hierarchical Crew, CrewAI Hierarchical Task

Компонент **CrewAI Hierarchical Crew** представляет группу агентов и управляет их совместной работой в иерархической структуре. Компонент позволяет создать команду с менеджером, контролирующим выполнение задач. Для получения дополнительной информации см. документацию по CrewAI Hierarchical Crew.

Компонент принимает следующие параметры:

Название	Отображаемое название	Описание
agents	Агенты	Входной параметр. Список объектов агентов — членов команды.
tasks	Задачи	Входной параметр. Список объектов HierarchicalTask для выполнения.
manager_llm	LLM менеджера	Входной параметр. Языковая модель для агента-менеджера.
manager_agent	Агент-менеджер	Входной параметр. Конкретный агент, выступающий в роли менеджера.
verbose	Подробный вывод	Входной параметр. Включает подробный вывод для детального журналирования.
memory	Память	Входной параметр. Конфигурация памяти для команды.

Название	Отображаемое название	Описание
use_cache	Использовать кэш	Входной параметр. Включает кэширование результатов.
max_rpm	Максимальный RPM	Входной параметр. Максимальное количество запросов в минуту.
share_crew	Общий доступ к команде	Входной параметр. Определяет, будет ли информация о команде общедоступна между агентами.
function_calling_llm	LLM для вызова функций	Входной параметр. Языковая модель для вызова функций.
crew	Crew	Выходной параметр. Сконструированный объект Crew с иерархическим выполнением задач.

CrewAI Sequential Crew, CrewAI Sequential Task

Компонент **CrewAI Sequential Crew** представляет группу агентов, задачи которых выполняются последовательно. Компонент позволяет создать команду, выполняющую задачи в определённом порядке. Для получения дополнительной информации см. документацию по CrewAI Sequential Crew.

Компонент принимает следующие параметры:

Название	Отображаемое название	Описание
tasks	Задачи	Входной параметр. Список объектов SequentialTask для выполнения.
verbose	Подробный вывод	Входной параметр. Включает подробный вывод для детального журналирования.
memory	Память	Входной параметр. Конфигурация памяти для команды.
use_cache	Использовать кэш	Входной параметр. Включает кэширование результатов.
max_rpm	Максимальный RPM	Входной параметр. Максимальное количество запросов в минуту.
share_crew	Общий доступ к команде	Входной параметр. Определяет, будет ли

Название	Отображаемое название	Описание
		информация о команде общедоступна между агентами.
function_calling_llm	LLM для вызова функций	Входной параметр. Языковая модель для вызова функций.
crew	Crew	Выходной параметр. Сконструированный объект Crew с последовательным выполнением задач.

CrewAI Sequential Task Agent

Этот компонент создаёт задачу CrewAI и связанного с ней агента, позволяя определять последовательные задачи с конкретными ролями и возможностями агентов. Для получения дополнительной информации см. документацию по CrewAI Sequential Task.

Компонент принимает следующие параметры:

Название	Отображаемое название	Описание
role	Роль	Входной параметр. Роль агента.
goal	Цель	Входной параметр. Цель агента.
backstory	Предыстория	Входной параметр. Предыстория агента.
tools	Инструменты	Входной параметр. Инструменты, доступные агенту.
llm	Языковая модель	Входной параметр. Языковая модель, управляющая агентом.
memory	Память	Входной параметр. Определяет, будет ли у агента память.
verbose	Подробный вывод	Входной параметр. Включает подробный вывод.
allow_delegation	Разрешить делегирование	Входной параметр. Определяет, может ли агент делегировать задачи другим агентам.
allow_code_execution	Разрешить выполнение кода	Входной параметр. Определяет, может ли агент выполнять код.
agent_kwargs	Дополнительные настройки агента	Входной параметр. Дополнительные

Название	Отображаемое название	Описание
task_description	Описание задачи	именованные аргументы для агента. Входной параметр. Текст с описанием цели и порядка выполнения задачи.
expected_output	Ожидаемый результат	Входной параметр. Чёткое определение ожидаемого результата задачи.
async_execution	Асинхронное выполнение	Входной параметр. Флаг, указывающий на асинхронное выполнение задачи.
previous_task	Предыдущая задача	Входной параметр. Предыдущая задача в последовательности для цепочки.
task_output	Последовательные задачи	Выходной параметр. Список объектов SequentialTask — созданных задач.

Пакет эмбеддингов

- **Сходство эмбеддингов:** заменено встроенной функцией поиска по схожести в компонентах векторного хранилища.
- **Эмбеддер текста:** заменено компонентами модели встраивания.

Пакет векторных хранилищ

Этот пакет содержит только устаревший компонент **Локальная БД**. Все остальные компоненты векторных хранилищ можно найти в соответствующих пакетах поставщиков, например в пакете **DataStax**.

Локальная БД

Замените компонент **Локальная БД** компонентом векторного хранилища **Chroma DB** (из пакета **Chroma**) или другим компонентом векторного хранилища.

Компонент **Локальная БД** читает и записывает данные в постоянный, находящийся в памяти экземпляр Chroma DB, предназначенный для использования с модулем «Генеративный ИИ». Он поддерживает отдельные режимы чтения и записи, автоматическое управление коллекциями и по умолчанию сохраняет данные в каталоге кэша Генеративный ИИ.

Задайте параметр **Режим** в соответствии с нужной операцией, затем настройте остальные параметры. Некоторые параметры доступны только в одном режиме.

Чтобы создать локальное векторное хранилище Chroma или записать в него данные, используйте режим **Загрузка**.

Параметры режима **Загрузка**:

Название	Тип	Описание
Название коллекции (collection_name)	Строка	Входной параметр. Имя коллекции векторного хранилища Chroma. По умолчанию: astra. Доступно только в режиме Загрузка .
Каталог для сохранения (persist_directory)	Строка	Входной параметр. Базовый каталог для создания и хранения векторного хранилища. При использовании компонента в нескольких потоках или с несколькими коллекциями данные сохраняются по пути \$PERSISTENT_DIRECTORY/vector_stores/\$COLLECTION_NAME. Если не указано, используется каталог конфигурации Генеративный ИИ. Подробнее см. в разделе Управление памятью.
Эмбединги (embedding)	Эмбединги	Входной параметр. Функция встраивания для использования в векторном хранилище.
Разрешить дубликаты (allow_duplicates)	Булево	Входной параметр. Если true (по умолчанию), новые записи не проверяются на дубликаты. Если false, документы, совпадающие с уже существующими в коллекции, не добавляются. Дедупликация применяется ко всей коллекции или к числу записей, указанному в параметре limit. Доступно только в режиме Загрузка .
Загружаемые данные (ingest_data)	Data или DataFrame	Входной параметр. Записи для сохранения в коллекцию. Записи встраиваются и индексируются для

Название	Тип	Описание
		семантического поиска. Доступно только в режиме Загрузка .
Лимит (limit)	Целое	Входной параметр. Максимальное количество записей для сравнения, когда параметр Разрешить дубликаты установлен в false. Ограничение лимита может повысить производительность при записи в большие коллекции, но допускает появление отдельных дублирующихся записей. Доступно только в режиме Загрузка .
Чтобы читать из локального векторного хранилища Chroma, используйте режим Поиск .		
Параметры режима Поиск :		
Название	Тип	Описание
Каталог для сохранения (persist_directory)	Строка	Входной параметр. Базовый каталог векторного хранилища. При использовании компонента в нескольких потоках или с несколькими коллекциями данные хранятся по пути \$PERSISTENT_DIRECTORY/vector_stores/\$COLLECTION_NAME. Если не указано, используется каталог конфигурации Генеративный ИИ. Подробнее см. в разделе Управление памятью.
Существующие коллекции (existing_collections)	Строка	Входной параметр. Выберите ранее созданную коллекцию для поиска. Доступно только в режиме Поиск .
Эмбединги (embedding)	Эмбединги	Входной параметр. Функция встраивания для использования в векторном хранилище.

Название	Тип	Описание
Тип поиска (search_type)	Строка	Входной параметр. Тип поиска: Similarity или MMR. Доступно только в режиме Поиск .
Поисковый запрос (search_query)	Строка	Входной параметр. Запрос для поиска по схожести. Доступно только в режиме Поиск .
Количество результатов (number_of_results)	Целое	Входной параметр. Количество возвращаемых результатов поиска. По умолчанию: 10. Доступно только в режиме Поиск .

Пакет Zep

Память Zep Chat

Компонент **Память Zep Chat** является устаревшим. Замените его на компонент **История сообщений**.

Этот компонент создаёт экземпляр ZepChatMessageHistory, обеспечивая хранение и извлечение сообщений чата с использованием Zep — сервера памяти для LLM.

Компонент принимает следующие параметры:

Название	Тип	Описание
url	MessageText	Входной параметр. URL-адрес экземпляра Zep. Обязательный.
api_key	SecretString	Входной параметр. Ключ API для аутентификации с экземпляром Zep.
api_base_path	Dropdown	Входной параметр. Версия API. Варианты: api/v1 или api/v2.
session_id	MessageText	Входной параметр. Уникальный идентификатор сессии чата. Необязательный.
message_history	BaseChatMessageHistory	Выходной параметр. Экземпляр ZepChatMessageHistory для сессии.

26.2 Agentics

Пакет компонентов Agentics использует LLM для преобразования табличных данных. Добавляйте или заполняйте столбцы построчно с помощью компонента **aMap**, объединяйте

несколько строк в одну с помощью компонента **aReduce** или генерируйте синтетические строки с помощью компонента **aGenerate**.

Определите структуру генерируемых данных в таблице **Схема** компонентов. Например:

Столбец	Тип	Описание	Обязательно
Название	String	Имя выходного поля	Да
Type	Выпадающий список	str, int, float, bool или dict	Да
Описание	String	Что представляет собой это поле и как оно должно генерироваться	Нет
As List	Boolean	Если true, поле представляет собой список указанного типа (например, list[str])	Нет

Все компоненты Agentic возвращают **DataFrame**.

Компонент aGenerate

aGenerate генерирует синтетические данные из схемы или на основе примера DataFrame. Используйте его для создания тестовых данных, аугментации данных или примеров для документации.

Например, следующее определение схемы создаёт вывод DataFrame:

Определение схемы:

- customer_name (str): Полное имя
- email (str): Адрес электронной почты
- age (int): Возраст от 18 до 80
- purchase_categories (str, As List): Список категорий продуктов, которые были куплены

Вывод DataFrame (пример, сгенерировано 10 строк):

customer_name	email	age	purchase_categories
Sarah Johnson	sarah.j@email.com	34	Electronics, Books, Home & Garden
Michael Chen	m.chen@email.com	28	Sports, Clothing, Electronics
...	...	...	...

Параметры

Название	Тип	Описание
Языковая модель	Выпадающий список	Выберите провайдера LLM и модель.
Ввод DataFrame	DataFrame	Необязательно. Пример DataFrame для обучения; используются только

Название	Тип	Описание
Схема	Таблица	первые 50 строк. Если не указан, используется Схема. Определите столбцы для генерации, если Ввод DataFrame не предоставлен. См. определение схемы компонента.
Инструкции	String	Необязательные инструкции для генерации.
Количество строк	Integer	Сколько синтетических строк создать. По умолчанию: 10.

Компонент aMap

aMap преобразует каждую строку входных данных с помощью инструкций на естественном языке и заданной схемы вывода (одна строка на входе — одна строка на выходе). Используйте **aMap** для обогащения данных столбцами, сгенерированными LLM, такими как тональность, категории и извлечение сущностей.

Строки обрабатываются параллельно; размер пакета по умолчанию — 10. Потребление токенов масштабируется с количеством строк.

Например, **aMap** сохраняет каждую входную строку и заполняет sentiment, confidence и key_topics с помощью подключённого LLM.

Ввод DataFrame:

review_id	text
1	Great product, fast shipping!
2	Terrible quality, broke after one use

Определение схемы:

- sentiment (str): “positive”, “negative” или “neutral”
- confidence (float): Оценка уверенности 0–1
- key_topics (str, As List): Основные упомянутые темы

Вывод DataFrame:

review_id	text	sentiment	confidence	key_topics
1	Great product, fast shipping!	positive	0.95	product quality, shipping
2	Terrible quality, broke after one use	negative	0.92	quality, durability

Параметры

Название	Тип	Описание
Языковая модель	Выпадающий список	Выберите провайдера LLM и модель.
Ввод DataFrame	DataFrame	Входной DataFrame (список словарей или DataFrame). Каждая строка

Название	Тип	Описание
		обрабатывается независимо.
Схема	Таблица	Определите структуру и типы для генерируемых столбцов. См. определение схемы компонента.
Инструкции	String	Инструкции на естественном языке для преобразования каждой строки в схему вывода.
As List	Boolean	Если true, генерировать несколько экземпляров схемы для каждой строки и объединять их.
Keep Source Columns	Boolean	Если true, добавить новые столбцы к исходным данным; если false, вернуть только сгенерированные столбцы. Игнорируется, если As List равно true. По умолчанию: true.

Компонент aReduce

aReduce агрегирует все строки входного DataFrame в одну строку согласно схеме вывода. Используйте **aReduce** для создания резюме, отчётов или консолидированных выводов. Чтобы агрегировать строки в список, установите As List в значение true в компоненте.

Все строки отправляются в одном запросе; потребление токенов может быть высоким для больших DataFrame. Рассмотрите возможность предварительной фильтрации или выборки данных.

Например, **aReduce** берёт все входные строки и создаёт одну строку с агрегатами, сгенерированными LLM: суммирует доход в total_revenue, определяет самый продаваемый продукт в best_selling_product и формирует краткое summary о продажах.

Ввод DataFrame:

date	product	revenue	units
2024-01-01	Widget A	1200	50
2024-01-02	Widget B	800	30
2024-01-03	Widget A	1500	60

Определение схемы:

- total_revenue (float): Сумма всех доходов
- best_selling_product (str): Продукт с наибольшим количеством единиц
- summary (str): Краткое резюме на естественном языке

Вывод DataFrame:

total_revenue	best_selling_product	summary
3500	Widget A	Over 3 days, Widget A was the best seller with 110

total_revenue	best_selling_product	summary units, generating \$2700 in revenue.
Параметры		
Название	Тип	Описание
Языковая модель	Выпадающий список	Выберите провайдера LLM и модель.
Ввод DataFrame	DataFrame	Входной DataFrame (список словарей или DataFrame). Обязательно.
Схема	Таблица	Определите структуру и типы для агрегированного вывода. См. определение схемы компонента.
As List	Boolean	Если true, вывод представляет собой список экземпляров схемы.
Инструкции	String	Необязательные инструкции для агрегирования. Если не указаны, LLM выводит их из описаний полей.

Производительность и устранение неполадок

- **Потребление токенов:** aMap масштабируется с количеством строк; aReduce отправляет все строки за один вызов; aGenerate масштабируется с количеством экземпляров. Используйте меньшие пакеты или выборку из больших наборов данных, чтобы снизить затраты.
- **Размер пакета:** По умолчанию 10; максимум 25. Большие пакеты повышают пропускную способность, но увеличивают задержку на пакет.
- **Ошибки:** Если Agentics не найден, выполните `uv pip install agentics-py==0.3.1` и перезапустите Генеративный ИИ. При ошибках API или ключа установите API-ключ провайдера как глобальную переменную или переменную окружения. При ошибках DataFrame убедитесь, что ввод представляет собой список словарей, либо используйте вывод компонента DataFrame.

26.3 AI/ML API

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **AI/ML**.

Генерация текста через AI/ML API

Этот компонент создаёт экземпляр модели ChatOpenAI с использованием AI/ML API. Вывод — исключительно **Языковая модель** (LanguageModel), которую можно подключить к другому LLM-управляемому компоненту, например к компоненту **Умное преобразование**.

Для получения дополнительной информации см. документацию AI/ML API по интеграции с модулем «Генеративный ИИ» и компоненты языковых моделей.

Параметры генерации текста через AI/ML API

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для неограниченного количества токенов. Диапазон: 0-128000.
model_kwargs	Dictionary	Входной параметр. Дополнительные именованные аргументы для модели.
model_name	String	Входной параметр. Имя модели AIML, которую следует использовать. Параметры предопределены в AIML_CHAT_MODELS.
aiml_api_base	String	Входной параметр. Базовый URL API AIML. По умолчанию https://api.aimlapi.com .
api_key	SecretString	Входной параметр. API-ключ AIML для использования с моделью.
temperature	Float	Входной параметр. Управляет случайностью вывода. По умолчанию: 0.1.

Эмбединги AI/ML API

Компонент **Эмбединги AI/ML API** генерирует эмбединги с использованием AI/ML API.

Вывод — Embeddings, а именно экземпляр AIMLEmbeddingsImpl.

Для получения дополнительной информации об использовании компонентов моделей эмбедингов в потоках см. компоненты моделей эмбедингов.

Параметры эмбедингов AI/ML API

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
model_name	String	Входной параметр. Имя модели эмбедингов AI/ML,

Название	Тип	Описание
		которую следует использовать.
aiml_api_key	SecretString	Входной параметр. API-ключ для аутентификации в сервисе AI/ML.

26.4 ALTK

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

ALTK содержит функции, называемые *компонентами*. Они отличаются от компонентов Генеративный ИИ. Все компоненты ALTK доступны через компонент Генеративный ИИ **ALTK Агент**. **ALTK Агент** реализует компоненты из Agent Lifecycle Toolkit. Функции ALTK можно включать или отключать независимо.

- **Предварительная проверка инструмента:** Проверяет вызовы инструментов перед выполнением, чтобы убедиться в их уместности и корректности, используя компонент рефракции SPARC. Эта проверка предотвращает выполнение агентами некорректных вызовов инструментов.
- **Обработка JSON после вызова инструмента:** Обрабатывает большие JSON-ответы инструментов, генерируя Python-код на лету для извлечения релевантных данных. Это помогает уменьшить размер контекста и улучшает способность агента работать с большими ответами инструментов, особенно при работе с API, возвращающими объёмные JSON-данные. Компонент выводит сообщение с ответом агента, которое передаётся следующему компоненту в потоке вместо полного JSON, который уже не нужен.

Для получения дополнительной информации см. документацию Agent Lifecycle Toolkit.

Для примера компонента ALTK в модуле «Генеративный ИИ» см. видеоруководство **ALTK in Генеративный ИИ: надёжная обработка JSON-ответов в вашем AI-агенте**.

Параметры ALTK Агент

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
agent_llm	Выпадающий список	Входной параметр. Провайдер модели, который агент использует для генерации ответов.
enable_tool_validation	Булево	Входной параметр. Если включено, вызовы инструментов проверяются через SPARC перед выполнением на предмет уместности и корректности. По умолчанию: true.

Название	Тип	Описание
enable_post_tool_reflection	Булево	Входной параметр. Если включено, выводы инструментов автоматически обрабатываются через JSON-обработчик, когда вывод является JSON и превышает пороговый размер. По умолчанию: true.
response_processing_size_threshold	Integer	Входной параметр. Постобработка вывода инструмента выполняется только если длина ответа превышает этот порог в символах. По умолчанию: 100. Расширенный параметр.
tools	List[Tool]	Входной параметр. Список инструментов, доступных агенту.
system_prompt	String	Входной параметр. Системный промпт, предоставляющий контекст агенту.
input_value	String	Входной параметр. Пользовательский ввод для агента.
memory	Memory	Входной параметр. Память, которую агент использует для сохранения контекста.
max_iterations	Integer	Входной параметр. Максимальное количество итераций, разрешённых агенту.
verbose	Булево	Входной параметр. Определяет, следует ли выводить промежуточные шаги агента.
handle_parsing_errors	Булево	Входной параметр. Определяет, следует ли обрабатывать ошибки разбора в агенте.

26.5 Amazon

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Amazon**.

Amazon Bedrock Converse

Этот компонент генерирует текст с использованием Amazon Bedrock LLMs через Bedrock Converse API.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel). В частности, вывод **Языковая модель** представляет собой экземпляр ChatBedrockConverse, настроенный в соответствии с параметрами компонента.

Используйте вывод **Языковая модель**, когда необходимо применить модель Amazon Bedrock в качестве LLM для другого компонента, основанного на LLM, например компонента **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. компоненты языковых моделей.

Параметры Amazon Bedrock Converse

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
input_value	Строка	Входной параметр. Входная строка для генерации текста.
system_message	Строка	Входной параметр. Системное сообщение, передаваемое модели.
stream	Булево	Входной параметр. Включить потоковую передачу ответа. Работает только в чате. По умолчанию: false.
model_id	Строка	Входной параметр. Модель Amazon Bedrock, которую следует использовать.
aws_access_key_id	SecretString	Входной параметр. Идентификатор ключа доступа AWS для аутентификации. Обязательно.
aws_secret_access_key	SecretString	Входной параметр. Секретный ключ доступа AWS для аутентификации. Обязательно.

Название	Тип	Описание
aws_session_token	SecretString	Входной параметр. Токен сессии для вашей учётной записи AWS. Требуется только для временных учётных данных.
credentials_profile_name	Строка	Входной параметр. Имя профиля учётных данных AWS, который следует использовать. Если не указано, будет использован профиль по умолчанию.
region_name	Строка	Входной параметр. Регион AWS, где находятся ваши ресурсы Bedrock. По умолчанию: us-east-1.
endpoint_url	Строка	Входной параметр. Пользовательский URL конечной точки для сервиса Bedrock.
temperature	Float	Входной параметр. Управляет случайностью вывода. Более высокие значения делают вывод более случайным. По умолчанию: 0.7.
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. По умолчанию: 4096.
top_p	Float	Входной параметр. Параметр ядерной выборки (nucleus sampling). Управляет разнообразием вывода. По умолчанию: 0.9.
top_k	Integer	Входной параметр. Ограничивает количество токенов с наивысшей вероятностью, которые учитываются при выборке. Примечание: не все модели поддерживают top_k. По умолчанию: 250.

Название	Тип	Описание
disable_streaming	Булево	Входной параметр. Если true, отключает потоковую передачу ответов. Полезно для пакетной обработки. По умолчанию: false.
additional_model_fields	Словарь	Входной параметр. Дополнительные специфичные для модели параметры для тонкой настройки поведения.

Эмбединги Amazon Bedrock

Компонент **Эмбединги Amazon Bedrock** используется для загрузки моделей векторного представления из Amazon Bedrock.

Для получения дополнительной информации об использовании компонентов моделей векторного представления в потоках см. компоненты моделей эмбедингов.

Параметры эмбедингов Amazon Bedrock

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
model_id	String	Входной параметр. Идентификатор модели для вызова, например amazon.titan-embed-text-v1. Эквивалентен свойству modelId в API list-foundation-models.
aws_access_key_id	SecretString	Входной параметр. Идентификатор ключа доступа AWS для аутентификации.
aws_secret_access_key	SecretString	Входной параметр. Секретный ключ доступа AWS для аутентификации.
aws_session_token	SecretString	Входной параметр. Токен сессии для вашей учётной записи AWS.
credentials_profile_name	String	Входной параметр. Имя профиля учётных данных AWS в ~/.aws/credentials или ~/.aws/config, содержащего ключи доступа или информацию о роли.

Название	Тип	Описание
region_name	String	Входной параметр. Регион AWS для использования, например us-west-2. При отсутствии используется переменная окружения AWS_DEFAULT_REGION или регион, указанный в ~/.aws/config.
endpoint_url	String	Входной параметр. URL для указания конкретной конечной точки сервиса, отличной от стандартной конечной точки AWS.

Загрузчик в корзину S3

Компонент **Загрузчик в корзину S3** загружает файлы в корзину Amazon S3. Он предназначен для обработки входных данных типа Data от компонента **Чтение файла** или **Папка**. Если вы передаёте данные Data из других компонентов, протестируйте результаты перед запуском потока в продакшене.

Требуется пакет boto3, который включён в вашу установку Генеративный ИИ.

Компонент генерирует журналы, но не передаёт данные в поток.

Параметры загрузчика в корзину S3

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Идентификатор ключа доступа AWS	SecretString	Входной параметр. Идентификатор ключа доступа AWS для аутентификации.
Секретный ключ доступа AWS	SecretString	Входной параметр. Секретный ключ доступа AWS для аутентификации.
Название корзины	String	Входной параметр. Имя корзины S3, в которую будут загружаться файлы.
Стратегия загрузки файлов	String	Входной параметр. Стратегия загрузки файлов. Сохранить данные (по умолчанию) — перебирает входные данные Data, записывает путь к файлу и текстовое содержимое, и

Название	Тип	Описание
		загружает каждый файл в указанную корзину S3, если доступны и путь к файлу, и содержимое. Сохранить исходный файл — перебирает список входных данных, извлекает путь к файлу из каждого элемента данных, загружает файл в указанную корзину S3, если путь доступен, и записывает путь к загружаемому файлу.
Входные данные	Data	Входной параметр. Входные данные Data, которые следует перебрать и загрузить в виде файлов в указанную корзину S3.
Префикс S3	String	Входной параметр. Необязательный префикс (путь к папке) внутри корзины S3, куда будут загружаться файлы.
Удалить путь	Boolean	Входной параметр. Удалять ли путь к файлу при загрузке. По умолчанию: false.

Устаревшие компоненты Amazon

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, но рекомендуется заменить их поддерживаемыми компонентами как можно скорее. Рекомендуемые замены указаны в баннере **Устаревшее** на компонентах в ваших потоках. Они также указаны в примечаниях к выпуску и документации Генеративный ИИ, когда это возможно.

Если вы не уверены, как заменить устаревший компонент, воспользуйтесь **Поиском** по провайдеру, сервису или названию компонента. Компонент мог быть объявлен устаревшим в пользу полностью нового компонента, похожего компонента или новой версии того же компонента в другой категории.

Если явной замены нет, рассмотрите возможность адаптации другого компонента под ваш сценарий. Например, многие **основные компоненты** предоставляют общую функциональность, которая может поддерживать несколько провайдеров и сценариев использования, — такие как компонент **API-запрос**.

Если ни один из этих вариантов не подходит, вы можете использовать код устаревшего компонента для создания собственного пользовательского компонента или начать обсуждение об этом устаревшем компоненте.

Чтобы не поощрять использование устаревших компонентов в новых потоках, эти компоненты по умолчанию скрыты. В визуальном редакторе вы можете нажать **Настройки компонента**, чтобы переключить фильтр **Устаревшее**.

Следующие компоненты Amazon находятся в статусе устаревших:

Amazon Bedrock

Компонент **Amazon Bedrock** был объявлен устаревшим в пользу компонента **Amazon Bedrock Converse**, который использует Bedrock Converse API для обработки диалогов.

Чтобы использовать модели Amazon Bedrock в ваших потоках, используйте вместо этого компонент **Amazon Bedrock Converse**.

26.6 Anthropic

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Anthropic**.

Для получения дополнительной информации о функциях и возможностях Anthropic, используемых компонентами Anthropic, см. документацию Anthropic.

Генерация текста Anthropic

Компонент **Anthropic** генерирует текст с использованием чат-моделей и языковых моделей Anthropic, таких как Claude.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel). В частности, вывод **Языковая модель** представляет собой экземпляр ChatAnthropic, настроенный в соответствии с параметрами компонента.

Используйте вывод **Языковая модель**, когда необходимо использовать модель Anthropic в качестве LLM для другого компонента, управляемого LLM, например компонента **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. компоненты языковых моделей.

Параметры генерации текста Anthropic

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для неограниченного количества токенов. По умолчанию: 4096.
model	String	Входной параметр. Имя модели Anthropic, которую следует использовать. Варианты включают различные модели Claude 3.

Название	Тип	Описание
anthropic_api_key	SecretString	Входной параметр. Ваш API-ключ Anthropic для аутентификации.
temperature	Float	Входной параметр. Управляет случайностью в выводе. По умолчанию: 0.1.
anthropic_api_url	String	Входной параметр. Конечная точка Anthropic API. По умолчанию https://api.anthropic.com , если не указано.
prefill	String	Входной параметр. Предварительный текст для направления ответа модели.

26.7 Apify

Apify — это платформа для веб-скрейпинга и извлечения данных, предоставляющая более 3 000 готовых облачных инструментов, называемых **Actors**.

Ваши потоки могут использовать компонент **Акторы Apify** для запуска **Actors** и выполнения таких задач, как извлечение данных, анализ контента и операции с SQL.

Использование компонента «Акторы Apify» в потоке

- Добавьте компонент **Акторы Apify** в ваш поток и настройте его следующим образом:
 - Токен Apify:** Введите ваш Apify API token.
 - Актор:** Введите ID актора, который хотите запустить из Apify Actor Store. Например, у Website Content Crawler ID актора — `apify/website-content-crawler`.
 - Входные данные запуска:** Введите JSON-данные для настройки запуска актора.
 - Настройте дополнительные параметры в зависимости от выбранного актора и вашего сценария использования.
- Подключите компонент к другим компонентам в вашем потоке. Компонент может использоваться для выполнения задач как отдельный шаг в потоке или как инструмент для агента.

Чтобы включить **режим инструмента** для этого компонента, измените тип вывода компонента с **Вывод** на **Инструмент**, а затем подключите его к порту **Инструменты** компонента **Агент**.

Компоненты **Акторы Apify** выводят результаты выполнения актора в виде JSON-объекта в типе Data Генеративный ИИ.

Примеры потоков с «Акторами Apify»

Ниже приведены несколько сценариев использования компонентов **Акторы Apify** в модуле «Генеративный ИИ»:

- Извлечь текстовый контент сайта в формате Markdown:** используйте Website Content Crawler Actor для извлечения текстового контента в формате Markdown с сайта, а затем подключите **Вывод** к входу компонента **Парсер** для дальнейшей обработки.

- **Обработать веб-контент с помощью агента:** прикрепите Website Content Crawler Actor к компоненту **Агент** в качестве инструмента, чтобы агент мог самостоятельно решать, извлекать ли контент сайта на основе ввода из чата. Агент может взять извлечённые данные и преобразовать их в резюме, инсайты или структурированные ответы, делая информацию более пригодной для использования.
- **Анализировать профили в социальных сетях с помощью нескольких акторов:** выполните комплексное исследование социальных сетей с использованием нескольких акторов Arify:
 - Используйте Google Search Results Scraper Actor для поиска релевантных профилей в социальных сетях.
 - Используйте TikTok Data Extractor Actor для сбора данных и видео.
 - Прикрепите оба актора в качестве инструментов к компоненту **Агент**, чтобы собрать ссылки из Google и контент из TikTok, а затем проанализировать данные и предоставить инсайты о человеке, бренде или теме.

26.8 arXiv

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **arXiv**.

Поиск arXiv

Этот компонент ищет и извлекает статьи с arXiv.org.

Он возвращает список результатов поиска в виде DataFrame.

Параметры поиска arXiv

Название	Type	Описание
search_query	String	Входной параметр. Запрос поиска для статей arXiv. Например, quantum computing.
search_type	String	Входной параметр. Поле, в котором выполнять поиск.
max_results	Integer	Входной параметр. Максимальное количество возвращаемых результатов.

26.9 AssemblyAI

Компоненты AssemblyAI позволяют применять мощные модели Speech AI к вашему приложению для задач, таких как:

- Транскрибирование аудио- и видеофайлов
- Форматирование транскриптов
- Создание субтитров
- Применение LLM к аудиофайлам

Для получения дополнительной информации о функциях и возможностях AssemblyAI, используемых компонентами AssemblyAI, см. AssemblyAI API Docs.

Требования

- Учётная запись AssemblyAI и API-ключ AssemblyAI.
Введите ключ в поле *AssemblyAI Ключ API* во всех компонентах Генеративный ИИ, требующих ключ AssemblyAI.
- Необязательно: для использования LeMUR необходима платная учётная запись AssemblyAI, так как LeMUR не включён в бесплатный аккаунт.

Компоненты

AssemblyAI запуск транскрипции

Этот компонент позволяет отправлять аудио- или видеофайл для транскрипции.

Подсказка: Вы можете зафиксировать путь этого компонента, чтобы отправлять файл только один раз.

- **Ввод:**
 - AssemblyAI Ключ API: Ваш API-ключ.
 - Аудиофайл: Аудио- или видеофайл для транскрипции.
 - Модель распознавания речи (Опционально): Выберите класс моделей. По умолчанию *Best*. См. *speech models* для получения дополнительной информации.
 - Автоматическое определение языка (Опционально): Включить автоматическое определение языка.
 - Язык (Опционально): Язык аудиофайла. Может быть установлен вручную, если автоматическое определение языка отключено. См. *supported languages* для списка поддерживаемых кодов языков.
 - Определение дикторов (Опционально): Обнаружить говорящих в аудиофайле и определить, что каждый из них сказал.
 - Ожидаемое количество дикторов (Опционально): Установить ожидаемое количество говорящих, если включено определение дикторов.
 - URL аудиофайла (Опционально): URL аудио- или видеофайла для транскрипции. Может использоваться вместо *Аудиофайл*.
 - Пунктуация (Опционально): Применять пунктуацию. По умолчанию *true*.
 - Форматирование текста (Опционально): Применять регистр и форматирование текста. По умолчанию *true*.
- **Вывод:**
 - ID транскрипции: Идентификатор транскрипции.

Опрос статуса транскрипции AssemblyAI

Этот компонент позволяет опрашивать транскрипции. Он проверяет статус транскрипции каждые несколько секунд, пока транскрипция не будет завершена.

- **Ввод:**
 - AssemblyAI Ключ API: Ваш API-ключ.
 - Интервал опроса (Опционально): Интервал опроса в секундах. По умолчанию 3.
- **Вывод:**
 - Результат транскрипции: JSON-ответ AssemblyAI завершённой транскрипции. Содержит текст и другую информацию.

AssemblyAI Получение субтитров

Этот компонент позволяет генерировать субтитры в формате SRT или VTT.

- **Ввод:**

- AssemblyAI Ключ API: Ваш API-ключ.
- Результат транскрипции: Вывод компонента **Стенограмма**.
- Формат субтитров: Формат субтитров (SRT или VTT).
- Символов на субтитр (Опционально): Максимальное количество символов в субтитре (0 — без ограничения).
- **Вывод:**
 - Субтитры: JSON-ответ с полем subtitles, содержащим субтитры в формате SRT или VTT.

AssemblyAI LeMUR

Этот компонент позволяет применять большие языковые модели к устным данным с использованием AssemblyAI LeMUR framework.

LeMUR автоматически загружает транскрипцию в качестве дополнительного контекста, упрощая применение LLM к аудиоданным. Вы можете использовать его для задач, таких как суммирование аудио, извлечение инсайтов или задавание вопросов.

- **Ввод:**
 - AssemblyAI Ключ API: Ваш API-ключ.
 - Результат транскрипции: Вывод компонента **Стенограмма**.
 - Промпт: Текст для подачи запроса модели. Вы можете ввести запрос в этом поле или подключить его к компоненту **Шаблон промпта**.
 - Финальная модель: Модель, используемая для финального запроса после выполнения сжатия. По умолчанию — Claude 3.5 Sonnet.
 - Температура (Опционально): Температура, используемая для модели. По умолчанию 0.0.
 - Максимальный размер вывода (Опционально): Максимальный размер вывода в токенах, до 4000. По умолчанию 2000.
 - Конечная точка (Опционально): Конечная точка LeMUR для использования. По умолчанию — “task”. Для “summary” и “question-answer” ввод промпта не требуется. См. LeMUR API docs для получения дополнительной информации.
 - Вопросы (Опционально): Список вопросов, разделённых запятыми. Используется только если *Конечная точка* равна “question-answer”.
 - ID транскрипций (Опционально): Список идентификаторов транскриптов, разделённых запятыми. LeMUR может выполнять действия над несколькими транскриптами. Если указано, *Результат транскрипции* игнорируется.
- **Вывод:**
 - LeMUR Ответ: Сгенерированный ответ LLM.

AssemblyAI Список транскрипций

Этот компонент может использоваться как отдельный компонент для отображения всех ранее созданных транскриптов.

- **Ввод:**
 - AssemblyAI Ключ API: Ваш API-ключ.
 - Лимит (Опционально): Максимальное количество транскриптов для получения. По умолчанию 20, используйте 0 для всех.
 - Фильтр (Опционально): Фильтрация по статусу транскрипта.
 - Дата создания (Опционально): Получать только транскрипты, созданные в указанную дату (YYYY-MM-DD).

- Только ограниченные (Опционально): Получать только ограниченные транскрипты, переопределяя фильтр статуса.
- **Вывод:**
 - Список транскриптов: Список всех транскриптов с информацией, такой как идентификатор транскрипта, статус и данные.

Процесс потока

1. Пользователь вводит аудио- или видеофайл.
2. Пользователь также может ввести промпт для LLM. В этом примере мы хотим сгенерировать резюме транскрипта.
3. Поток отправляет аудиофайл на транскрипцию.
4. Поток проверяет статус транскрипции каждые несколько секунд, пока транскрипция не будет завершена.
5. Поток разбирает результат транскрипции и выводит транскрибированный текст.
6. Поток также генерирует субтитры.
7. Поток применяет промпт LLM для генерации резюме.
8. В качестве отдельного компонента можно отобразить все транскрипты.

Запуск потока транскрипции и Speech AI

1. Создайте поток вручную или импортируйте готовый JSON-файл:
 - Рекомендуется: Скачать JSON-файл потока AssemblyAI Transcription and Speech AI
 - Создайте пустой поток, а затем добавьте ранее описанные компоненты в ваш поток, соединяя их, как показано на диаграмме потока.
2. Введите ваш API-ключ AssemblyAI во всех компонентах, требующих ключ (**Начать транскрипцию, Стенограмма, Получить субтитры, LeMUR, Список стенограмм**).
3. Выберите аудио- или видеофайл для компонента **Начать транскрипцию**.
 Опционально: После добавления файла в компонент **Начать транскрипцию** запустите и зафиксируйте компонент, чтобы отправлять файл только один раз, независимо от количества запусков потока. Для этого нажмите **Запустить компонент**, чтобы предварительно загрузить файл, затем нажмите **Показать больше** и выберите **Зафиксировать**, чтобы зафиксировать результат. Последующие запуски потока используют кэшированный вывод зафиксированного компонента.
4. Протестируйте транскрипцию, нажав **Запустить компонент** на компоненте **Парсер**.
 Убедитесь, что указанный шаблон имеет вид {text}.
 Запуск одного компонента запускает все предшествующие компоненты, а также выбранный компонент, после чего выполнение потока останавливается. В данном случае компоненты **Начать транскрипцию** и **Стенограмма** находятся выше по потоку относительно компонента **Парсер**. Если вы зафиксировали компонент **Начать транскрипцию**, поток передаст кэшированный вывод из **Начать транскрипцию** и запустит компонент **Стенограмма**, чтобы получить результат транскрипции. Проверьте журналы потока или изучите вывод компонента **Парсер**, чтобы увидеть транскрибированный текст.
5. Чтобы сгенерировать субтитры и запустить полный поток, нажмите **Запустить компонент** на компоненте **Список стенограмм**.

Настройка

Поток можно настроить следующим образом:

1. Изменение параметров в компоненте **Начать транскрипцию**.
2. Изменение формата субтитров в компоненте **Получить субтитры**.
3. Изменение промпта LLM в компоненте **LeMUR**.
4. Изменение параметров LLM (например, температуры) в компоненте **LeMUR**.

Устранение неполадок

Если возникли проблемы:

1. Убедитесь, что API-ключ правильно установлен во всех компонентах, требующих ключ.
2. Для использования LeMUR необходимо перейти на платный аккаунт AssemblyAI, так как LeMUR не включён в бесплатный аккаунт.
3. Проверьте, что все компоненты правильно соединены в потоке.
4. Просмотрите журналы Генеративный ИИ на наличие сообщений об ошибках.
5. Ознакомьтесь с документацией AssemblyAI API.
6. Свяжитесь с службой поддержки AssemblyAI.

26.10 Azure

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Azure**.

Azure OpenAI

Этот компонент генерирует текст с использованием Azure OpenAI LLMs.

Он может возвращать либо **Модель Ответа** (Message), либо **Языковую модель** (LanguageModel). Конкретно, вывод **Языковая модель** представляет собой экземпляр AzureChatOpenAI, настроенный согласно параметрам компонента.

Используйте вывод **Языковая модель**, когда необходимо использовать модель Azure OpenAI в качестве LLM для другого LLM-управляемого компонента, например компонента **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. Компоненты Языковой модели.

Параметры Azure OpenAI

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Type	Описание
Название модели	String	Входной параметр. Указывает имя модели Azure OpenAI, используемой для генерации Текст.
Конечная точка Azure	String	Входной параметр. Ваш конечный пункт Azure, включая ресурс.
Название развертывания	String	Входной параметр. Указывает имя развертывания.

Название	Type	Описание
Версия API	String	Входной параметр. Указывает версию API Azure OpenAI, которая будет использоваться.
Ключ API	SecretString	Входной параметр. Ваш ключ API Azure OpenAI.
Температура	Float	Входной параметр. Указывает температуру сэмплирования. По умолчанию 1.0.
Максимальное количество токенов	Integer	Входной параметр. Указывает максимальное количество токенов для генерации. По умолчанию 1000.
Входное значение	String	Входной параметр. Указывает входные Текст для генерации Текст.
Потоковая передача	Boolean	Входной параметр. Указывает, следует ли потоково передавать ответ модели. По умолчанию false.

Azure OpenAI Эмбединги

Компонент **Azure OpenAI Эмбединги** генерирует эмбединги с использованием моделей Azure OpenAI.

Для получения дополнительной информации об использовании компонентов моделей эмбедингов в потоках см. Компонент Модели Эмбедингов.

Параметры Azure OpenAI Эмбединги

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Type	Описание
Модель	String	Входной параметр. Имя модели для использования. По умолчанию: text-embedding-3-small.
Конечная точка Azure	String	Входной параметр. Ваш конечный пункт Azure, включая ресурс, например https://example-resource.azure.openai.com/ .
Название развертывания	String	Входной параметр. Имя развертывания.

Название	Type	Описание
Версия API	String	Входной параметр. Версия API для использования, варианты включают различные даты.
Ключ API	String	Входной параметр. Ключ API, необходимый для доступа к сервису Azure OpenAI.

26.11 Baidu

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Baidu**.

Qianfan

Компонент **Qianfan** генерирует Текст с использованием языковых моделей Qianfan.

Он может выдавать либо **Модель Ответа** (Message), либо **Языковая модель** (LanguageModel).

Используйте вывод **Языковая модель**, когда хотите использовать модель Qianfan в качестве LLM для другого компонента, управляемого LLM, например, компонента **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. Язык компоненты моделей и документацию Qianfan.

26.12 Bing

Bundles содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Bing**.

API поиска Bing

Этот компонент позволяет вызвать API Bing Поиск.

Он возвращает список результатов поиска в виде DataFrame.

параметры API поиска Bing

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
bing_subscription_key	SecretString	Входной параметр. Ключ подписки Bing API.
input_value	String	Входной параметр. Ввод поискового запроса.
bing_search_url	String	Входной параметр. Пользовательский URL Bing Поиск.

Название	Тип	Описание
k	Integer	Входной параметр. Количество возвращаемых результатов поиска.

26.13 Cassandra

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Cassandra**, включая компоненты, которые читают и записывают данные в кластеры Apache Cassandra, такие как OSS Cassandra и базы данных Astra DB.

Cassandra векторное хранилище

Используйте компонент **Cassandra** для чтения или записи в векторное хранилище на основе Cassandra с помощью экземпляра `CassandraVectorStore`.

Об экземплярах векторного хранилища

Поскольку Генеративный ИИ построен на основе LangChain, компоненты векторного хранилища используют экземпляр `LangChain vector store` для реализации базовых функций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента, такими как строка подключения, имя индекса и схема.

В коде компонента это часто создаётся как `vector_store`, но некоторые компоненты векторного хранилища используют другое имя, например имя поставщика.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от поставщика эти параметры могут использовать значения по умолчанию или позволять изменение через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Для получения информации о конкретных параметрах см. справочник API LangChain и документацию поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса вашего вектора База данных, он возвращает результаты поиска, которые можно передать последующим компонентам в вашем потоке в виде списка объектов `Data` или табличного `DataFrame`. Если поддерживаются оба типа, формат можно задать рядом с выводом компонента векторного хранилища Порт в визуальном редакторе.

Для руководства по использованию вектора База данных в потоке см. Создать чат-бот с векторным RAG.

Параметры Cassandra

Вы можете просмотреть параметры компонента векторного хранилища, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора определённых опций для других параметров. Условные параметры могут не отображаться на панели **Управление**, пока не заданы необходимые зависимости.

Для получения информации о допустимых значениях и функциональности см. Векторный поиск в Cassandra или изучите код компонента.

Название	Type	Описание
База данных_ref	String	Входной параметр. Точки подключения для База данных или ID Astra База данных.
username	String	Входной параметр. Имя пользователя для базы данных. Оставьте пустым для Astra DB.
token	SecretString	Входной параметр. Пароль пользователя для База данных или токен приложения Astra.
keyspace	String	Входной параметр. Имя keyspace, содержащего векторное хранилище, указанное в Имя таблицы (table_name).
table_name	String	Входной параметр. Имя таблицы или коллекции, являющейся векторным хранилищем.
ttl_seconds	Integer	Входной параметр. Время жизни (time-to-live) добавленных текстов, если поддерживается кластером. Имеет значение только для записей.
batch_size	Integer	Входной параметр. Количество записей для обработки в одной партии.
setup_mode	String	Входной параметр. Режим конфигурации для настройки таблицы Cassandra.
cluster_kwargs	Dict	Входной параметр. Дополнительные аргументы (keyword arguments) для кластера Cassandra.
search_query	String	Входной параметр. Запрос строка для поиска по сходству. Имеет значение только для чтения.
ingest_data	Данные	Входной параметр. Данные для загрузки в

Название	Type	Описание
		векторное хранилище в виде необработанных фрагментов и эмбеддингов. Имеет значение только для записей.
embedding	Эмбеддинги	Входной параметр. Функция Эмбеддинги для использования.
number_of_results	Integer	Входной параметр. Количество результатов для возврата в поиске. Имеет значение только для чтения.
search_type	String	Входной параметр. Тип поиска, который следует выполнить. Имеет значение только для чтения.
search_score_threshold	Float	Входной параметр. Минимальный порог сходства (score) для результатов поиска. Имеет значение только для чтения.
search_filter	Dict	Входной параметр. Необязательный словарь фильтров метаданных для поиска, применяемый в дополнение к векторному поиску. Имеет значение только для чтения.
body_search	String	Входной параметр. Документ текстовые поисковые запросы. Имеет значение только для чтения.
enable_body_search	Boolean	Входной параметр. Флаг для включения поиска по содержимому. Имеет значение только для чтения.

Cassandra Память чата

Компонент **Cassandra Память Чата** извлекает и сохраняет сообщения чата, используя базу данных на основе Apache Cassandra.

Память чата передаётся между компонентами хранения памяти в виде типа данных Memory. Конкретно, компонент создаёт экземпляр CassandraChatMessageHistory, который является классом истории сообщений чата LangChain и использует База данных Cassandra для хранения.

Для получения дополнительной информации об использовании внешней памяти чата в потоках см. компонент **История сообщений**.

Параметры Cassandra Память чата

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
База данных_ref	MessageText	Входной параметр. Точки подключения к База данных Cassandra или идентификатору Astra DB База данных. Обязательно.
username	MessageText	Входной параметр. Имя пользователя для Cassandra. Оставьте пустым для Astra DB.
token	SecretString	Входной параметр. Пароль для Cassandra или токен для Astra DB. Обязательно.
keyspace	MessageText	Входной параметр. Пространство ключей (keyspace) в Cassandra или пространство имён в Astra DB. Обязательно.
table_name	MessageText	Входной параметр. Имя таблицы или коллекции для хранения сообщений. Обязательно.
session_id	MessageText	Входной параметр. Уникальный идентификатор чат-сессии. Необязательно.
cluster_kwargs	Dictionary	Входной параметр. Дополнительные именованные аргументы для конфигурации кластера Cassandra. Необязательно.

Cassandra Граф

Компонент **Cassandra Граф** использует `CassandraGraphVectorStore`, экземпляр `LangChain graph vector store`, для обхода графа и поиска документов на основе графа в совместимом кластере на базе Cassandra. Он также поддерживает запись в векторное хранилище.

Если вы используете компонент векторного хранилища для запроса вашего вектора База данных, он генерирует результаты поиска, которые можно передать последующим компонентам в вашем потоке в виде списка объектов `Data` или табличного `DataFrame`. Если поддерживаются оба типа, вы можете задать формат рядом с выводом компонента векторного хранилища Порт в визуальном редакторе.

Cassandra Граф параметры

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображать Название	Информация
База данных_ref	Контактные лица / Идентификатор базы данных Astra	Входной параметр. Контактные точки для База данных или ID Astra База данных. Обязательно.
username	Имя пользователя	Входной параметр. Имя пользователя для базы данных. Оставьте пустым для Astra DB.
token	Пароль / Токен Astra DB	Входной параметр. Пароль пользователя для База данных или токен приложения Astra. Обязательно.
keyspace	Пространство ключей	Входной параметр. Имя keyspace, содержащего векторное хранилище, указанное в Имя таблицы (table_name). Обязательно.
table_name	Имя таблицы	Входной параметр. Имя таблицы или коллекции, являющейся векторным хранилищем. Обязательно.
setup_mode	Режим настройки	Входной параметр. Режим конфигурации для настройки таблицы Cassandra. Варианты: Sync (по умолчанию) или Off.
cluster_kwargs	Кластерные аргументы	Входной параметр. Необязательный словарь дополнительных аргументов для кластера Cassandra.
search_query	Поисковый запрос	Входной параметр. Строка запроса для поиска по схожести. Применяется только при чтении.
ingest_data	Загружаемые данные	Входной параметр. Данные для загрузки в векторное хранилище в виде необработанных фрагментов и эмбеддингов.

Название	Отображать Название	Информация
		Применяется только при записи.
embedding	Эмбединги	Входной параметр. Модель эмбединга для использования.
number_of_results	Количество результатов	Входной параметр. Количество результатов, возвращаемых при поиске по схожести. Применяется только при чтении. По умолчанию: 4.
search_type	Тип поиска	Входной параметр. Тип поиска для использования. Варианты: Traversal (по умолчанию), MMR Traversal, Similarity, Similarity with score threshold или MMR (Max Marginal Relevance).
depth	Глубина прохождения	Входной параметр. Максимальная глубина ребер для обхода. Применяется только если Тип поиска (search_type) установлен в Traversal или MMR Traversal. По умолчанию: 1.
search_score_threshold	Порог оценки поиска	Входной параметр. Минимальный порог схожести для результатов поиска. Применяется только при чтении с типом поиска Similarity with score threshold.
search_filter	Фильтр метаданных поиска	Входной параметр. Необязательный словарь фильтров метаданных для применения вместе с обходом графа и поиском по схожести.

Смотрите также

- Пакет **DataStax**

26.14 Chroma

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Chroma**.

Chroma DB

Вы можете использовать компонент **Chroma DB** для чтения и записи в Chroma База данных с помощью экземпляра Chroma векторного хранилища. Поддерживает удалённые или в-памяти экземпляры с сохранением или без него.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для реализации базовых функций чтения и записи. Эти экземпляры зависят от провайдера и настраиваются в соответствии с параметрами компонента, такими как строка подключения, имя индекса и схема.

В коде компонента это часто создаётся как `vector_store`, но некоторые компоненты векторных хранилищ используют другое имя, например имя провайдера.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от провайдера эти параметры могут иметь значения по умолчанию или быть изменяемыми через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Для получения информации о конкретных параметрах см. справочник API LangChain и документацию провайдера векторного хранилища.

При записи компонент может создать новую базу данных или коллекцию в указанном месте.

Эфемерное (непостоянное) локальное векторное хранилище Chroma удобно для тестирования потоков поиска по векторам, когда нет необходимости сохранять базу данных. Если вы используете компонент векторного хранилища для запроса вашей векторной базы данных, он возвращает результаты поиска, которые можно передать последующим компонентам в вашем потоке в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, вы можете задать формат рядом с выводом Порт компонента векторного хранилища в визуальном редакторе.

Использовать компонент Chroma DB в потоке

В следующем примере потока используется один компонент **Chroma DB** как для чтения, так и для записи:

- При записи он разбивает Data из компонента **URL** на фрагменты, вычисляет эмбединги с помощью подключённого компонента **Модель эмбедингов**, а затем загружает фрагменты и эмбединги в векторное хранилище Chroma. Чтобы выполнить запись, нажмите **Запустить компонент** на компоненте **Chroma DB**.
- При чтении он использует ввод чата для выполнения поиска по схожести в векторном хранилище, а затем выводит результаты поиска в чат. Чтобы выполнить чтение, откройте **Playground** и введите сообщение в чат.

После однократного запуска потока вы можете нажать **Проверить Вывод** на каждом компоненте, чтобы понять, как данные преобразуются при передаче от компонента к компоненту.

Параметры Chroma DB

Вы можете просмотреть параметры компонента векторного хранилища, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора конкретных опций для других параметров. Условные параметры могут не отображаться на панели **Управление**, пока не будут заданы необходимые зависимости.

Для получения информации о допустимых значениях и функциональности см. документацию провайдера или просмотрите код компонента.

Название	Тип	Описание
Название коллекции (collection_name)	Строка	Входной параметр. Имя вашей коллекции векторов Chroma. По умолчанию: Генеративный ИИ.
Каталог для сохранения (persist_directory)	Строка	Входной параметр. Чтобы сохранить Chroma БД, укажите относительный или абсолютный путь к каталогу для хранения файла chroma.sqlite3. Оставьте пустым для эфемерной базы данных. При чтении или записи в существующую БД укажите путь к постоянному каталогу.
Загружаемые данные (ingest_data)	Данные или DataFrame	Входной параметр. Data или DataFrame ввод, содержащий записи для записи в векторное хранилище. Применимо только для записей.
Поисковый запрос (search_query)	Строка	Входной параметр. Запрос, используемый для векторного поиска. Применимо только для чтения.
Cache Векторное хранилище (cache_vector_store)	Булево	Входной параметр. Если true, компонент кэширует векторное хранилище в памяти для более быстрых чтений. По

Название	Тип	Описание
Эмбединги (embedding)	Эмбединги	умолчанию: Включено (true). Входной параметр. Функция встраивания для использования в векторном хранилище. По умолчанию Chroma DB использует встроенную модель встраивания, либо можно подключить компонент Модель эмбедингов для использования другого провайдера или модели.
CORS Allow Origins (chroma_server_cors_allow_origins)	Строка	Входной параметр. Разрешённые источники CORS для сервера Chroma.
Chroma Хост сервера (chroma_server_host)	Строка	Входной параметр. Хост сервера Chroma.
Chroma HTTP-порт сервера (chroma_server_http_port)	Целое	Входной параметр. HTTP Порт сервера Chroma.
Chroma Серверный порт gRPC (chroma_server_grpc_port)	Целое	Входной параметр. gRPC Порт сервера Chroma.
Chroma Сервер SSL включен (chroma_server_ssl_enabled)	Булево	Входной параметр. Включить SSL для сервера Chroma.
Разрешить дубликаты (allow_duplicates)	Булево	Входной параметр. Если true (по умолчанию), записи не проверяют наличие дубликатов в коллекции, позволяя хранить несколько копий одинакового контента. Если false, записи не добавят документы, совпадающие с уже существующими в коллекции. Если false, можно строго применять дедупликацию, просматривая всю коллекцию или только количество записей, указанное в limit.

Название	Тип	Описание
Тип поиска (search_type)	Строка	Применимо только для записей. Входной параметр. Тип поиска, который следует выполнить, либо Similarity, либо MMR. Применимо только для чтения.
Количество результатов (number_of_results)	Целое	Входной параметр. Количество результатов поиска для возврата. По умолчанию: 10. Применимо только для чтения.
Лимит (limit)	Целое	Входной параметр. Лимит количество записей для сравнения, когда Разрешить дубликаты равно false. Это может улучшить производительность при записи в большие коллекции, но может привести к появлению некоторых дублирующих записей. Применимо только для записей.

26.15 Cleanlab

Cleanlab добавляет автоматизацию и доверие к каждому входному элементу данных и каждому предсказанию, получаемому от решений AI и RAG.

Используйте компоненты Cleanlab для интеграции оценок Cleanlab с модулем «Генеративный ИИ» и откройте надёжные агентные, RAG и LLM конвейеры с набором оценок и исправлений Cleanlab.

Вы можете использовать эти компоненты, чтобы количественно оценить надёжность любого ответа LLM с оценкой от 0 до 1 и объяснить, почему ответ может быть хорошим или плохим. Для конвейеров RAG или агентных с контекстом вы можете оценивать достаточность контекста, обоснованность, полезность и ясность запроса с помощью количественных оценок. Кроме того, вы можете исправлять ответы с низким уровнем доверия, выдавая предупреждения или запасные ответы.

Для аутентификации требуется ключ API Cleanlab.

Оценщик Cleanlab

Компонент **Cleanlab Evaluator** оценивает и объясняет надёжность пары запрос-ответ с использованием Cleanlab. Для получения дополнительной информации о том, как работает оценка, см. документацию Cleanlab.

Параметры Оценщика Cleanlab

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Type	Описание
system_prompt	Сообщения	Входной параметр. Системное сообщение, добавляемое перед запросом. Необязательно.
prompt	Сообщения	Входной параметр. Ввод, видимый пользователю, для LLM.
response	Сообщения	Входной параметр. Ответ модели для оценки.
cleanlab_api_key	Secret	Входной параметр. Ваш ключ API Cleanlab.
cleanlab_evaluation_model	Dropdown	Входной параметр. Модель оценки, используемая Cleanlab, например GPT-4 или Claude. Не обязательно должна быть той же моделью, которая сгенерировала ответ.
quality_preset	Dropdown	Входной параметр. Приоритет между скоростью оценки и точностью.

Выходные данные Evaluator Cleanlab

Компонент **Cleanlab Evaluator** имеет три возможных выхода.

Название	Type	Описание
score	number, float	Отображает доверительный балл от 0 до 1.
explanation	Message	Предоставляет объяснение доверительного балла.
response	Message	Возвращает оригинальный ответ для удобного соединения с компонентом Cleanlab Remediator .

Cleanlab Remediator

Компонент **Cleanlab Remediator** использует показатель доверия из **Cleanlab Evaluator** component, чтобы определить, показывать ли ответ, выдавать предупреждение или заменять ответ LLM.

Этот компонент имеет параметры для порога оценки, предупреждения Текст и сообщения по умолчанию, которые можно настроить по необходимости.

Выводом является **Remediated Ответ** (remediated_response), который представляет собой Message, содержащий окончательное сообщение, отображаемое пользователю после применения логики исправления.

Параметры Cleanlab Remediator

Название	Тип	Описание
response	Сообщения	Входной параметр. Ответ, который может быть исправлен.
score	Number	Входной параметр. Оценка доверия из CleanlabEvaluator.
explanation	Сообщения	Входной параметр. Объяснение, которое добавляется, если показывается предупреждение. Необязательно.
threshold	Float	Входной параметр. Минимальная оценка доверия, при которой ответ остаётся без изменений.
show_untrustworthy_response	Boolean	Входной параметр. Показывать ли оригинальный ответ с предупреждением или скрывать его, если ответ считается недостоверным.
untrustworthy_warning_text	Запрос	Входной параметр. Предупреждающий Текст для недостоверных ответов.
fallback_text	Запрос	Входной параметр. Сообщение по умолчанию, если ответ скрыт.

Оценщик Cleanlab RAG

Компонент **Cleanlab RAG Evaluator** оценивает выводы конвейера RAG и LLM с точки зрения доверия, достаточности контекста, обоснованности ответа, полезности и простоты запроса, используя метрики оценки Cleanlab.

Вы можете сочетать этот компонент с компонентом **Cleanlab Remediator** для исправления ответов с низким уровнем доверия, полученных из конвейера RAG.

Параметры оценщика Cleanlab RAG

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
cleanlab_api_key	Секрет	Входной параметр. Ваш ключ API Cleanlab.
cleanlab_evaluation_model	Выпадающий список	Входной параметр. Модель оценки, используемая Cleanlab, например GPT-4 или Claude. Не обязательно, чтобы это была та же модель, которая сгенерировала ответ.
quality_preset	Выпадающий список	Входной параметр. Компромисс между скоростью оценки и точностью.
context	Сообщения	Входной параметр. Полученный контекст из вашей RAG-системы.
query	Сообщения	Входной параметр. Исходный запрос пользователя.
response	Сообщения	Входной параметр. Ответ модели, основанный на контексте и запросе.
run_context_sufficiency	Булево	Входной параметр. Оценить, поддерживает ли контекст ответ на запрос.
run_response_groundedness	Булево	Входной параметр. Оценить, обоснован ли ответ контекстом.
run_response_helpfulness	Булево	Входной параметр. Оценить полезность ответа.
run_query_ease	Булево	Входной параметр. Оценить, является ли запрос расплывчатым, сложным или враждебным.

Выводы оценщика Cleanlab RAG

Компонент **Cleanlab RAG Evaluator** имеет следующие варианты вывода:

Название	Тип	Описание
trust_score	Число	Общий показатель доверия.
trust_explanation	Сообщения	Объяснение показателя доверия.
other_scores	Словарь	Словарь необязательных

Название	Тип	Описание
evaluation_summary	Сообщения	Сводка в формате Markdown, включающая запрос, контекст, ответ и результаты оценки.
response	Сообщения	Возвращает исходный ответ для удобного соединения с компонентом Cleanlab Remediator .

Пример потоков Cleanlab

Следующие примеры потоков показывают, как использовать компоненты **Cleanlab Evaluator** и **Cleanlab Remediator** для оценки и исправления ответов любого LLM, а также как использовать компонент **Cleanlab RAG Evaluator** для оценки выводов RAG-конвейера.

Оценка и исправление ответов от LLM

Этот поток оценивает и исправляет достоверность ответа от любого LLM с помощью компонентов **Cleanlab Evaluator** и **Cleanlab Remediator**.

Вы можете скачать JSON-файл потока оценки и исправления Или вы можете построить поток с нуля, подключив следующие компоненты:

- Подключите вывод Message из любого компонента **Языковая модель** или **Агент** к входу **Ответ** компонента **Cleanlab Evaluator**.
- Подключите компонент **Шаблон промпта** к входу **Запрос** компонента **Cleanlab Evaluator**.

При запуске потока компонент **Cleanlab Evaluator** возвращает оценку доверия и объяснение.

Компонент **Cleanlab Remediator** использует эту оценку доверия, чтобы решить, выводить оригинальный ответ, предупредить о нём или заменить его резервным ответом.

В этом примере показан ответ, который был определён как недостоверный (оценка .09) и помечен предупреждением компонентом **Cleanlab Remediator**.

Чтобы скрыть недостоверные ответы, настройте компонент **Cleanlab Remediator** на замену ответа резервным сообщением.

Оценка RAG-конвейера

В качестве примера создайте поток на основе шаблона **Векторное хранилище RAG**, а затем добавьте компонент **Cleanlab RAG Evaluator**, чтобы оценить контекст, запрос и ответ потока. Подключите выходы **контекст**, **запрос** и **ответ** из остальных компонентов RAG-потока к компоненту **Cleanlab RAG Evaluator**.

Вот пример вывода Evaluation Summary из компонента **Cleanlab RAG Evaluator**:

В Evaluation Summary содержатся запрос, контекст, ответ и все результаты оценки. В этом примере оценки Context Sufficiency и Response Groundedness низкие (оценка 0.002), поскольку контекст не содержит информации о запросе, а ответ не основан на контексте.

26.16 ClickHouse

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в **пакете ClickHouse**.

Хранилище векторов ClickHouse

Компонент **ClickHouse** читает и записывает данные в хранилище векторов ClickHouse, используя экземпляр ClickHouse хранилища векторов.

О экземплярах векторных хранилищ

Поскольку Генеративный ИИ построен на основе LangChain, компоненты хранилищ векторов используют экземпляр LangChain vector store для реализации базовых функций чтения и записи. Эти экземпляры зависят от провайдера и настраиваются в соответствии с параметрами компонента, такими как строка подключения, имя индекса и схема.

В коде компонента это часто создаётся как `vector_store`, однако некоторые компоненты хранилищ векторов используют другое название, например имя провайдера.

Некоторые классы LangChain не раскрывают все возможные опции в виде параметров компонента. В зависимости от провайдера эти опции могут иметь значения по умолчанию или быть изменяемыми через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Для получения информации о конкретных опциях см. справочник API LangChain и документацию провайдера хранилища векторов.

Если вы используете компонент векторного хранилища для запроса вашего вектора База данных, он возвращает результаты поиска, которые можно передать последующим компонентам в вашем потоке в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно задать рядом с выводом компонента хранилища векторов Порт в визуальном редакторе.

Для учебного примера использования вектора База данных в потоке см. Создать чат-бот с векторным RAG.

Параметры ClickHouse

Вы можете просмотреть параметры компонента хранилища векторов, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора определённых опций для других параметров. Условные параметры могут не отображаться на панели **Управление**, пока не будут заданы необходимые зависимости.

Для получения информации о допустимых значениях и функциональности см. ClickHouse Documentation или просмотрите component code.

Название	Отображать Название	Информация
host	Хост	Входной параметр. Имя хоста сервера ClickHouse. Обязательно. По умолчанию: localhost.
Порт	Порт	Входной параметр. Порт сервера ClickHouse. Обязательно. По умолчанию: 8123.

Название	Отображать Название	Информация
База данных	База данных	Входной параметр. Имя База данных ClickHouse. Обязательно.
table	Имя таблицы	Входной параметр. Имя таблицы ClickHouse. Обязательно.
username	Имя пользователя	Входной параметр. Имя пользователя ClickHouse для аутентификации. Обязательно.
password	Пароль	Входной параметр. Пароль ClickHouse для аутентификации. Обязательно.
index_type	index_type	Входной параметр. Тип индекса, либо annou (по умолчанию), либо vector_similarity.
Метрика	Метрика	Входной параметр. Метрики для вычисления расстояния при поиске по схожести. Варианты: angular (по умолчанию), euclidean, manhattan, hamming, dot.
secure	Use HTTPS/TLS	Входной параметр. Если true, включает HTTPS/TLS для сервера ClickHouse и переопределяет выводимые значения для аргументов интерфейса или Порт. По умолчанию: false.
index_param	Параметр индекса	Входной параметр. Параметры индекса. По умолчанию: 100,'L2Distance'.
index_query_params	Параметры запроса индекса	Входной параметр. Дополнительные параметры запросов индекса.
search_query	Поисковый запрос	Входной параметр. Строка запроса для поиска

Название	Отображать Название	Информация
ingest_data	Загружаемые данные	по схожести. Применяется только при чтении. Входной параметр. Записи для загрузки в векторное хранилище.
cache_vector_store	Cache хранилище	Векторное Входной параметр. Если true, компонент кэширует векторное хранилище в памяти для более быстрых чтений. По умолчанию: Включено (true).
embedding	Эмбединги	Входной параметр. Модель встраивания для использования.
number_of_results	Количество результатов	Входной параметр. Количество результатов поиска для возврата. По умолчанию: 4. Применяется только при чтении.
score_threshold	Пороговое значение оценки	Входной параметр. Порог для сравнения оценки схожести. По умолчанию: Не задано (без порога). Применяется только при чтении.

26.17 Cloudflare

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Cloudflare**.

Эмбединги Cloudflare Workers AI

Компонент **Эмбединги Cloudflare Workers AI** генерирует эмбединги (embeddings), используя модели Cloudflare Workers AI.

Для получения дополнительной информации об использовании компонентов моделей Эмбединги в потоках, см. Компоненты Модели эмбедингов.

Параметры Эмбедингов Cloudflare Workers AI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображать Название	Информация
account_id	Идентификатор учетной записи Cloudflare	Входной параметр. Ваш Идентификатор учетной записи Cloudflare.

Название	Отображать Название	Информация
api_token	Токен Cloudflare API	Входной параметр. Ваш Токен Cloudflare API.
model_name	Название модели	Входной параметр. Поддерживаемая модель для генерации эмбедингов.
strip_new_lines	Удалить новые строки	Входной параметр. Нужно ли удалять новые строки из входного текста.
batch_size	Размер батча	Входной параметр. Количество текстов для эмбединга в каждой партии.
api_base_url	Базовый URL-адрес Cloudflare API	Входной параметр. Базовый URL для API Cloudflare.
headers	Заголовки	Входной параметр. Дополнительные заголовки для запроса API генерации эмбедингов.

26.18 Cohere

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Cohere**.

Для получения дополнительной информации о функциях и возможностях Cohere, используемых в компонентах Cohere, см. документацию Cohere.

Cohere Генерация текста

Этот компонент генерирует текст с использованием языковых моделей Cohere.

Компонент может выводить либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте вывод **Языковая модель**, когда хотите применить модель Cohere в качестве LLM для другого компонента — например, **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. Компоненты языковой модели.

Параметры генерации текста Cohere

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Ввод	String	Входной текст для генерации ответа.
Системное сообщение	String	Системное сообщение для передачи модели.
Потоковая передача	Boolean	Определяет, следует ли использовать потоковую

Название	Тип	Описание
		передачу ответа. Работает только в режиме чата. По умолчанию: false.
API-ключ Cohere	SecretString	Ваш API-ключ Cohere.
Температура	Float	Степень случайности при выборке. Низкие значения (близкие к 0) дают более детерминированные результаты, высокие (близкие к 1) — более креативные. По умолчанию: 0.75.

Cohere Эмбединги

Компонент **Cohere Эмбединги** используется для загрузки моделей встраивания от Cohere.

Для получения дополнительной информации об использовании компонентов моделей встраивания в потоках см. Компоненты моделей эмбедингов.

Параметры Cohere Эмбединги

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
cohere_api_key	SecretString	API-ключ для аутентификации в сервисе Cohere.
model	String	Языковая модель для создания векторных представлений текстов документов и запросов. По умолчанию: embed-english-v2.0.
truncate	String	Способ обработки ввода, превышающего лимит токенов модели. Допустимые значения: NONE, START или END (по умолчанию). Подробнее см. в справочнике API Cohere по параметру truncate.
max_retries	Integer	Максимальное количество повторных попыток при неудачных запросах. По умолчанию: 3.

Название	Тип	Описание
user_agent	String	Строка User-Agent, включаемая в запросы. По умолчанию: langchain.
request_timeout	Float	Время ожидания ответа на запросы в секундах. По умолчанию: None.

Cohere Rerank

Этот компонент выполняет поиск и переранжирование документов с использованием API Cohere.

Выводит Data, содержащий переранжированные документы, количество которых ограничено параметром **Количество результатов**.

Параметры Cohere Rerank

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Поисковый запрос	String	Запрос для переранжирования документов.
Результаты поиска	Данные	Подключите вывод результатов поиска из компонента векторного хранилища. Используйте этот параметр для применения переранжирования после поиска по сходству в векторной базе данных.
Количество результатов	Integer	Количество документов, возвращаемых после переранжирования. По умолчанию: 3.
API-ключ Cohere	SecretString	Ваш ключ API Cohere.
Модель	String	Модель переранжирования для использования. По умолчанию: rerank-english-v3.0.

26.19 CometAPI

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **CometAPI**.

Для получения дополнительной информации о функциях и возможностях CometAPI см. документацию CometAPI.

CometAPI Генерация текста

Этот компонент генерирует текст с использованием языковых моделей CometAPI через совместимые с OpenAI конечные точки CometAPI.

Компонент может выводить либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте вывод **Языковая модель**, когда хотите применить модель CometAPI в качестве LLM для другого компонента — например, **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. Компоненты языковой модели.

Параметры CometAPI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
api_key	SecretString	Ключ API CometAPI для аутентификации.
model_name	String	Идентификатор модели CometAPI. Компонент автоматически получает список актуальных доступных моделей при наличии действительного API-ключа. Подробнее см. в документации CometAPI.
input_value	String	Входной текст для отправки модели.
system_message	String	Системное сообщение, задающее поведение ассистента.
max_tokens	Integer	Максимальное количество токенов для генерации. Установите 0 для снятия ограничения.
temperature	Float	Управляет случайностью вывода. Диапазон: [0.0, 2.0]. По умолчанию: 0.7.
seed	Integer	Значение для воспроизводимости результатов генерации.
model_kwargs	Dict	Дополнительные именованные аргументы для передачи модели.

Название	Тип	Описание
json_mode	Boolean	Если true, вывод будет в формате JSON независимо от наличия схемы.
stream	Boolean	Определяет, следует ли использовать потоковую передачу ответа. По умолчанию: false.
model	LanguageModel	Экземпляр ChatOpenAI, настроенный с параметрами CometAPI.

Использование CometAPI в потоке

1. Зарегистрируйтесь в аккаунте CometAPI.
2. Получите ваш API-ключ из панели управления CometAPI.
3. В Генеративный ИИ добавьте компонент **CometAPI** в поток.
4. Введите ваш API-ключ в поле **Ключ API CometAPI**.
5. Выберите предпочтительную модель из списка **Название модели**.
6. Настройте остальные параметры в соответствии с вашим сценарием использования.
7. Добавьте в поток необходимые компоненты.

Для базового теста добавьте компоненты **Ввод в чат** и **Вывод чата**, подключите их к компоненту **CometAPI** и нажмите **Playground**, чтобы проверить соединение и пообщаться с моделью.

Для более сложных сценариев подключите компонент **CometAPI** к другим компонентам — например, **Шаблон промпта**, **Агент** или **Умное преобразование**.

26.20 Composio

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Composio**.

Для получения информации о конкретных функциях Composio см. документацию Composio.

Использование компонентов Composio в потоке

Компоненты Composio в основном используются как инструменты для агентов.

Пакет **Composio** включает агрегированный компонент **Инструменты Composio** и следующие однослужебные компоненты:

Однослужебные компоненты Composio

- **AgentQL**
- **Agiled**
- **Airtable**
- **Apollo**
- **Asana**
- **Attio**
- **Bitbucket**
- **Bolna**
- **Brightdata**
- **Calendly**
- **Canva**
- **Canvas**

- **Coda**
- **Contentful**
- **Digicert**
- **Discord**
- **Dropbox**
- **ElevenLabs**
- **Exa**
- **Figma**
- **Finage**
- **Firecrawl**
- **Fireflies**
- **Fixer**
- **Flexisign**
- **Freshdesk**
- **GitHub**
- **Gmail**
- **Google BigQuery**
- **Google Calendar**
- **Google Classroom**
- **Google Docs**
- **Google Meet**
- **Google Sheets**
- **Google Задачи**
- **Heygen**
- **Instagram**
- **Jira**
- **Jotform**
- **Klaviyo**
- **Linear**
- **Listennotes**
- **Mem0**
- **Miro**
- **Missive**
- **Notion**
- **OneDrive**
- **Outlook**
- **Pandadoc**
- **PeopleDataLabs**
- **PerplexityAI**
- **Reddit**
- **SerpAPI**
- **Slack**
- **Slackbot**
- **Snowflake**
- **Supabase**
- **Tavily**
- **TimelinesAI**
- **Todoist**
- **Wrike**
- **YouTube**

Компонент **Инструменты Composio** служит точкой доступа к множеству сервисов Composio. Однако большинство этих сервисов также доступны как одноструженные компоненты, которые рекомендуется использовать вместо компонента **Инструменты Composio**.

Хотя одноструженные компоненты можно использовать для неагентных действий в потоках, они в основном применяются в **Режиме инструмента** с компонентами **Агент**. В отличие от них, компонент **Инструменты Composio** может *только* выступать в роли инструмента для агентов — неагентное использование не поддерживается.

Следующий пример демонстрирует, как использовать компонент **Composio Gmail** в качестве инструмента для компонента **Агент**. Это позволяет агенту при необходимости использовать функциональность Composio Gmail при генерации ответов. Другие компоненты Composio подключаются аналогичным образом.

1. Создайте поток на основе шаблона **Простой Агент**.
2. В **Пакетах** найдите пакет **Composio**, а затем добавьте компонент **Gmail** в поток.
3. В поле **Ключ API Composio** введите ваш API-ключ Composio или используйте глобальную переменную `COMPOSIO_API_KEY`. Подробнее см. раздел Аутентификация Composio.
Если ключ действителен, индикатор **Ошибка** заменяется индикатором **Успешно**, и список **Действия** заполняется действиями, доступными для вашего API-ключа.
4. В меню компонента включите **Режим инструмента**.
Если вы используете компонент **Инструменты Composio**, пропустите этот шаг — компонент уже настроен как инструмент.
5. В списке **Действия** настройте действия Gmail, которые вы хотите предоставить агенту. Вы можете выбрать нужные действия и отредактировать slug (метку) и описание каждого действия — это помогает агенту решить, какие инструменты использовать.
6. Подключите выход **Набора инструментов** компонента **Gmail** к входу **Инструменты** компонента **Агент**.
7. В компоненте **Агент** введите ваш API-ключ OpenAI или настройте компонент для использования другой языковой модели. Подробнее см. Компоненты языковой модели.
На данном этапе поток содержит четыре подключённых компонента:

- Компонент **Ввод в чат** подключён к порту **Ввод** компонента **Агент** — это позволяет запускать поток с помощью входящего запроса от пользователя или приложения.
- Компонент **Gmail** подключён к компоненту **Агент** в качестве инструмента — агент использует это подключение только если решит, что инструмент Gmail поможет ответить на запрос.
- Порт **Вывод** компонента **Агент** подключён к компоненту **Вывод чата**, который возвращает окончательный ответ пользователю или приложению.

8. Чтобы протестировать поток, нажмите **Playground** и спросите агента о доступных ему инструментах. Агент должен предоставить список подключённых инструментов, включая **Gmail** и встроенные инструменты компонента **Агент**.
9. Чтобы протестировать конкретный инструмент, попросите агента выполнить действие с его использованием. Например, попросите создать черновик письма:

Create a draft email with the subject line "Greetings from Composio"
recipient: "your.email@address.com"
Body content: "Hello from composio!"

В **Playground** отобразится логика работы агента при выборе инструмента `GMAIL_CREATE_EMAIL_DRAFT`. Пример сокращённого ответа:

The draft email with the subject "Greetings from Composio" and body "Hello from composio!" has been successfully created.

```
{
  "recipient_email": "your.email@address.com",
  "subject": "Greetings from Composio",
  "body": "Hello from composio!",
  "is_html": false
}
```

```
{
  "data": {
    "response_data": {
      "id": "r-237981011463568567",
      "message": {
        "id": "195dd80528171132",
        "threadId": "195dd80528171132",
        "labelIds": ["DRAFT"]
      }
    }
  },
  "error": null,
  "successful": true,
  "successful": true
}
```

Для дополнительного подтверждения вы можете зайти в аккаунт Gmail и найти сообщение в папке черновиков.

10. По желанию: чтобы добавить больше сервисов Composio, повторите описанные шаги для каждого дополнительного компонента. Для каждого из них укажите учётные данные, включите **Режим инструмента**, настройте действия и подключите к порту **Инструменты** компонента **Агент**.

Параметры Composio

Название	Тип	Описание
entity_id	String	Идентификатор сущности для аккаунта Composio. По умолчанию: default. Этот параметр скрыт по умолчанию в визуальном редакторе. Доступ к нему можно получить через панель инспекции компонента.
api_key	SecretString	Ключ API Composio для аутентификации на платформе Composio. Убедитесь, что ключ авторизует конкретный сервис, который вы хотите использовать. Подробнее см. раздел Аутентификация Composio.
tool_name	Connection	Только для компонента Инструменты Composio . Выберите сервис (инструмент) Composio для подключения.
action	List	Выберите действия для использования. Доступные действия зависят от сервиса.

Название	Тип	Описание
		Некоторые действия могут требовать премиум-доступ.

Аутентификация Composio

Компоненты Composio требуют аутентификации на платформе Composio с помощью ключа API Composio.

Ключ можно указать непосредственно в компонентах или использовать глобальную переменную `COMPOSIO_API_KEY`, которую Генеративный ИИ автоматически загружает из файла `.env`.

Ключ API Composio *только* обеспечивает соединение с платформой Composio. Аутентификация конкретного стороннего сервиса управляется через платформу Composio для каждого сервиса отдельно. Убедитесь, что ваш ключ API Composio предоставляет доступ к необходимым сервисам. Например, при использовании компонента **Composio Gmail** ключ должен иметь доступ к сервису Gmail.

Вывод Composio

При использовании в качестве инструментов для агента компоненты Composio выводят Tools — список инструментов для использования агентом. При вызове агентом ответ от сервиса Composio обрабатывается агентом и не передаётся напрямую пользователю или приложению.

В неагентных сценариях вывод представляет собой DataFrame, содержащий ответ от указанного сервиса Composio, в зависимости от используемого компонента и действия.

Поскольку компонент **Инструменты Composio** поддерживает *только* агентное использование, он не может выводить DataFrame. Все однослужебные компоненты Composio могут выводить как DataFrame, так и Tools.

26.21 Couchbase

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Couchbase**.

Векторное хранилище Couchbase

Компонент **Couchbase** читает и записывает данные в векторное хранилище Couchbase, используя экземпляр CouchbaseSearchVectorStore.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых функций чтения и записи. Эти экземпляры зависят от провайдера и настраиваются согласно параметрам компонента, таким как строка подключения, имя индекса и схема.

В коде компонента экземпляр обычно создаётся как `vector_store`, однако некоторые компоненты векторных хранилищ используют другое имя, например имя провайдера.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от провайдера эти параметры могут иметь значения по умолчанию или изменяться через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации провайдера векторного хранилища.

При использовании компонента векторного хранилища для запроса базы данных он генерирует результаты поиска, которые можно передать следующим компонентам в потоке в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат задаётся рядом с портом вывода компонента в визуальном редакторе.

Пошаговое руководство по использованию векторной базы данных в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры Couchbase

Просмотрите параметры компонента векторного хранилища, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Controls**, пока не заданы необходимые зависимости.

Сведения о допустимых значениях и функциональности см. в документации Couchbase или в коде компонента.

Название	Тип	Описание
couchbase_connection_string	SecretString	Строка подключения к кластеру Couchbase. Обязательно.
couchbase_username	String	Имя пользователя Couchbase для аутентификации. Обязательно.
couchbase_password	SecretString	Пароль Couchbase для аутентификации. Обязательно.
bucket_name	String	Название бакета Couchbase. Обязательно.
scope_name	String	Название области (scope) Couchbase. Обязательно.
collection_name	String	Название коллекции Couchbase. Обязательно.
index_name	String	Название индекса Couchbase. Обязательно.
ingest_data	Данные	Записи для загрузки в векторное хранилище. Применяется только к операциям записи.
search_query	String	Строка запроса для векторного поиска. Применяется только к операциям чтения.

Название	Тип	Описание
cache_vector_store	Boolean	Кэшировать ли векторное хранилище в памяти Генеративный ИИ для ускорения операций чтения. По умолчанию: включено (true).
embedding	Эмбединги	Функция встраивания (эмбедингов) для векторного хранилища.
number_of_results	Integer	Максимальное количество возвращаемых результатов поиска. По умолчанию: 4. Применяется только к операциям чтения.

26.22 CUGA

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Компонент **CUGA** поддерживает только модели **OpenAI** и **watsonx**. Чтобы использовать других провайдеров моделей, используйте основной компонент **Агент**. Компонент **CUGA (Configurable Generalist Agent)** — это продвинутый AI-агент, выполняющий сложные задачи с использованием инструментов, опциональной автоматизации браузера и генерации структурированного вывода.

Компонент **CUGA** может использоваться в потоках вместо компонента **Агент**. Как и основной компонент **Агент**, компонент **CUGA** может использовать инструменты, подключённые к его порту **Инструменты**, и сам может использоваться в качестве инструмента. Он также включает ряд дополнительных функций:

- Автоматизация браузера для веб-скрейпинга с помощью Playwright. Чтобы включить веб-скрейпинг, установите параметр `browser_enabled` компонента в значение `true`.
- Загрузка пользовательских инструкций для выполнения агентом. Чтобы использовать эту функцию, используйте вход компонента **Инструкции** для прикрепления Markdown-файлов, содержащих инструкции для агента.

Дополнительную информацию см. в репозитории проекта CUGA.

Использование компонента CUGA в потоке

Следующий пример демонстрирует изменение шаблона потока для использования компонента **CUGA**.

1. Создайте поток на основе шаблона **Простой Агент**, а затем замените компонент **Агент** на компонент **CUGA**.
2. Подключите компонент **Инструменты MCP** и компонент **Калькулятор** к порту **Инструменты** компонента **CUGA**, а затем подключите компонент **Инструменты MCP** к любому серверу MCP. В этом примере подключается сервер, содержащий данные о продажах для бизнес-организации.
3. Добавьте компонент **Чтение файла** и подключите его к порту **Инструкции** компонента **CUGA**. Либо нажмите **Редактировать текст**, чтобы открыть панель

Содержимое редактируемого текста, и введите инструкции непосредственно в компонент **CUGA**.

4. Создайте на компьютере файл Markdown с именем `instructions.md` и вставьте в него следующее содержимое. Важно чётко оформить документ с заголовками `## План` и `## Ответ`, чтобы агент правильно интерпретировал ваши инструкции.

`## План`

- Break down complex queries into subtasks
- Prioritize information gathering before execution
- Consider dependencies between actions
- Validate intermediate results before proceeding

`## Ответ`

- Provide concise summaries with key findings
- Include relevant data points and metrics
- Cite sources when using MCP tool results
- Use clear structure and formatting for readability

5. В компоненте **Чтение файла** нажмите **Добавить файл**, а затем прикрепите ваш файл `instructions.md`.

6. Откройте **Playground** и задайте агенту вопрос, для ответа на который может использоваться подключённый сервер MCP. В этом примере задаётся вопрос о данных продаж, предоставляемых сервером MCP, например *Какие типы учётных записей доступны?*. Агент описывает вызовы инструментов, которые он выполняет, и возвращает ответ в соответствии с инструкциями. Например, список доступных аккаунтов может быть очень большим, однако компонент **CUGA** возвращает краткое резюме согласно заданной политике ответа.

Based on the available data, here are the accounts:

Summit Solutions (NY) - Revenue: \$1,200,000
Pacific Ventures (CA) - Revenue: \$9,500,000
Lone Star Corp (TX) - Revenue: \$4,500,000
Mountain Peak Systems (CO) - Revenue: \$2,100,000
Digital Dynamics (CA) - Revenue: \$5,500,000
Cascade Computing (WA) - Revenue: \$4,300,000
Data Flow Systems (CA) - Revenue: \$8,900,000
Rocky Mountain Enterprises (CO) - Revenue: \$3,200,000
Blue Sky Partners (TX) - Revenue: \$500,000
Liberty Manufacturing (PA) - Revenue: \$3,400,000

This is a partial list; there are more accounts available. The total revenue across all accounts is \$210,200,000.

Параметры CUGA

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
<code>agent_llm</code>	Выпадающий список	Провайдер модели для агента.
<code>instructions</code>	Многострочная строка	Пользовательские инструкции, определяющие планирование и формат ответов агента. Можно передать напрямую или

Название	Тип	Описание
		через Markdown-файлы. Форматирование важно для корректной интерпретации агентом. См. раздел Использование компонента CUGA в потоке.
n_messages	Целое число	Количество сообщений истории чата для получения. Используется для поддержания контекста в текущих беседах, идентифицируемых по session_id. По умолчанию: 100.
format_instructions	Многострочная строка	Шаблон для структурированного вывода.
output_schema	Таблица	При наличии output_schema структурированные ответы проверяются по динамически построенной схеме. Некорректные элементы возвращаются вместе с ошибками валидации. Поля: name, description, type (str, int, float, bool, dict), multiple (в виде списка).
add_current_date_tool	Булево	Если true, добавляет инструмент, возвращающий текущую дату. По умолчанию: true.
lite_mode	Булево	Установите true, чтобы включить режим CugaLite для более быстрого выполнения при небольшом количестве инструментов. По умолчанию: true.
lite_mode_tool_threshold	Целое число	Порог для автоматического включения CugaLite. Если у компонента CUGA подключено меньше

Название	Тип	Описание
shortlisting_tool_threshold	Целое число	инструментов, чем указанный порог, CugaLite активируется автоматически. По умолчанию: 25.
d		Порог для предварительного отбора инструментов. Когда общее количество инструментов превышает этот порог, компонент CUGA включает функцию find_tools для фильтрации инструментов до меньшего набора перед принятием решений. Это снижает потребление токенов и повышает производительность при работе с большим количеством инструментов. По умолчанию: 35.
decomposition_strategy	Выпадающий список	Стратегия разложения задачи. flexible допускает несколько подзадач на одно приложение. exact требует одну подзадачу на одно приложение. По умолчанию: flexible.
browser_enabled	Булево	Включить встроенный браузер для веб-скрейпинга и поиска. Позволяет агенту использовать общий веб-поиск в своих ответах. Установите false, чтобы ограничить агента контекстом, предоставленным в потоке. По умолчанию: false.

26.23 DataStax

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **DataStax**, включая компоненты для чтения и записи данных в базы данных Astra DB.

Astra DB

Рекомендуется создать все необходимые базы данных, keyspace'ы и коллекции до настройки компонента **Astra DB**.

Вы можете создавать новые базы данных и коллекции через этот компонент, но только в визуальном редакторе Генеративный ИИ (не во время выполнения). Придётся подождать завершения инициализации базы данных или коллекции, прежде чем продолжить настройку потока. Кроме того, не все параметры конфигурации баз данных и коллекций доступны через компонент **Astra DB** — например, гибридный поиск, группы PCU, управление интеграцией vectorize и многорегиональные развёртывания. Компонент **Astra DB** читает и записывает данные в бессерверные базы данных Astra DB, используя экземпляр AstraDBVectorStore для вызовов Data API и DevOps API.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для реализации базовых функций чтения и записи. Эти экземпляры зависят от провайдера и настраиваются согласно параметрам компонента, таким как строка подключения, имя индекса и схема.

В коде компонента экземпляр обычно создаётся как vector_store, однако некоторые компоненты векторных хранилищ используют другое имя, например имя провайдера.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от провайдера эти параметры могут иметь значения по умолчанию или изменяться через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации провайдера векторного хранилища.

Параметры Astra DB

Просмотрите параметры компонента векторного хранилища, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Controls**, пока не заданы необходимые зависимости.

Сведения о допустимых значениях и функциональности см. в документации Astra DB Serverless или в коде компонента.

Название	Отображаемое название		Описание	
token	Токен	приложения	Токен	приложения
	Astra DB		Astra с правом доступа к вашей векторной базе данных. После проверки соединения дополнительные поля заполняются вашими существующими базами данных и коллекциями. Для создания базы данных через	

Название	Отображаемое название	Описание
		этот компонент должен иметь права администратора организации.
environment	Окружение	Окружение для конечной точки API Astra DB. Обычно prod.
database_name	База данных	Имя базы данных, к которой подключается этот компонент. Либо выберите New Database , чтобы создать новую базу данных, и дождитесь её инициализации перед настройкой остальных параметров.
endpoint	Конечная точка API Astra DB	Для многорегиональных баз данных выберите конечную точку API ближайшего дата-центра. Список регионов для многорегиональной базы данных см. в разделе List database regions. Поле автоматически заполняется при выборе базы данных и по умолчанию использует конечную точку основного региона.
keyspace	Пространство ключей	Keyspace в вашей базе данных, содержащий коллекцию из collection_name. По умолчанию: default_keyspace.
collection_name	Коллекция	Имя коллекции для использования в этом потоке. Либо выберите New Collection , чтобы создать новую коллекцию с ограниченными параметрами конфигурации. Для

Название	Отображаемое название	Описание
		корректной настройки провайдера встраивания и параметров поиска рекомендуется создать коллекцию в Astra Portal или через Data API до настройки этого компонента. Подробнее см. в разделе Manage collections in Astra DB Serverless.
embedding_model	Модель эмбедингов	Присоедините компонент модели эмбедингов для генерации векторных представлений. Доступно только если у указанной коллекции нет интеграции vectorize. При наличии интеграции vectorize компонент автоматически использует встроенную модель коллекции.
ingest_data	Загружаемые данные	Документы для загрузки в указанную коллекцию. Принимает тип Data или DataFrame.
search_query	Поисковый запрос	Строка запроса для векторного поиска.
cache_vector_store	Кэшировать векторное хранилище	Кэшировать ли векторное хранилище в памяти Генеративный ИИ для ускорения операций чтения. По умолчанию: включено (true).
search_method	Метод поиска	Метод поиска: Hybrid Search или Vector Search. Коллекция должна быть настроена для поддержки выбранного варианта. Значение по умолчанию зависит от возможностей коллекции. Все коллекции Astra DB Serverless с поддержкой векторов

Название	Отображаемое название	Описание
		поддерживают векторный поиск, однако гибридный поиск требует специальных настроек при создании коллекции. Подробнее см. в разделах Ways to find data in Astra DB Serverless и Create a collection that supports hybrid search.
reranker	Модель переранжирования	Модель переранжирования для гибридного поиска в зависимости от конфигурации коллекции. Доступно только для коллекций с поддержкой гибридного поиска. Чтобы проверить поддержку, получите метаданные коллекции и убедитесь, что поля lexical и rerank содержат "enabled": true.
lexical_terms	Лексические термины	Строка ключевых слов, разделённых пробелами, для гибридного поиска, например features data attributes characteristics. Доступно только для коллекций с поддержкой гибридного поиска. Подробнее см. в разделе Пример: гибридный поиск.
number_of_results	Количество результатов поиска	Количество возвращаемых результатов поиска. По умолчанию: 4.
search_type	Тип поиска	Тип поиска: Similarity (по умолчанию), Similarity with score threshold или MMR (Max Marginal Relevance).
search_score_threshold	Порог оценки поиска	Минимальный порог оценки сходства для результатов поиска при

Название	Отображаемое название	Описание
advanced_search_filter	Фильтр метаданных поиска	типе Similarity with score threshold. По умолчанию: 0. Необязательный словарь фильтров метаданных, применяемый дополнительно к векторному или гибриднему поиску.
autodetect_collection	Автоматическое определение коллекции	Получать ли список доступных коллекций автоматически после указания токена приложения и конечной точки API.
content_field	Поле содержимого	Для операций записи — название поля в документах, содержащего текст, для которого необходимо сгенерировать векторное представление.
deletion_field	Удаление по полю	При указании — документы в целевой коллекции с совпадающими метаданными удаляются перед загрузкой новых записей. Используйте эту настройку для операций upsert.
ignore_invalid_documents	Игнорировать некорректные документы	Игнорировать ли недействительные документы при записи. Если отключено (false), при обнаружении недействительного документа возникает ошибка. По умолчанию: включено (true).
astradb_vectorstore_kwargs	Параметры AstraDBVectorStore	Необязательный словарь дополнительных параметров для экземпляра AstraDBVectorStore.

Примеры Astra DB

Пример: векторный RAG

Пошаговое руководство по использованию векторных данных в потоке см. в разделе Создание чат-бота с векторным RAG. Следующий пример демонстрирует использование

компонентов векторного хранилища в потоках вместе с сопутствующими компонентами — моделью эмбедингов и языковой моделью. В примере рассматриваются важные детали конфигурации, функциональность и рекомендации по эффективному использованию компонентов. Пример не является исчерпывающим руководством для всех сценариев использования.

1. Создайте поток на основе шаблона **Векторное хранилище RAG**.

Шаблон содержит два подпотока: **Загрузка данных** загружает векторные представления и контент в векторную базу данных, а **Поисковый модуль** выполняет векторный поиск для получения релевантного контекста по запросу пользователя.

2. Настройте подключение к базе данных для обоих компонентов **Astra DB** или замените их другой парой компонентов векторного хранилища по вашему выбору. Убедитесь, что компоненты подключены к одному и тому же векторному хранилищу и что компонент в подпотоке **Поисковый модуль** может выполнять поиск по сходству.

Параметры каждого компонента векторного хранилища определяются его ролью в потоке. В этом примере подпоток **Загрузка данных** *записывает* в векторное хранилище, а подпоток **Поисковый модуль** *читает* из него. Параметры, связанные с поиском, актуальны только для компонента в подпотоке **Поисковый модуль**. Подробнее о конкретных параметрах см. в документации выбранного компонента.

3. Настройте модель эмбедингов одним из следующих способов:

- **Использовать модель OpenAI:** в обоих компонентах **OpenAI Эмбединги** введите ваш API-ключ OpenAI. Можно использовать модель по умолчанию или выбрать другую.
- **Использовать другого провайдера:** замените компоненты **OpenAI Эмбединги** другой парой компонентов модели эмбедингов и настройте параметры и учётные данные.
- **Использовать Astra DB vectorize:** при использовании векторного хранилища Astra DB с интеграцией vectorize удалите оба компонента **OpenAI Эмбединги** — интеграция автоматически сгенерирует векторные представления из **Загружаемых данных** (подпоток **Загрузка данных**) и **Поискового запроса** (подпоток **Поисковый модуль**).

Если векторное хранилище уже содержит векторные представления, убедитесь, что компоненты модели эмбедингов используют ту же модель, что применялась при их создании. Смешивание моделей эмбедингов в одном хранилище может привести к неточным результатам поиска. 4. Рекомендуется: в компоненте **Разделить текст** оптимизируйте параметры фрагментации под вашу модель эмбедингов. Например, если модель поддерживает не более 512 токенов, параметр **Размер фрагмента** не должен превышать это значение.

Поскольку подпоток **Поисковый модуль** передаёт пользовательский ввод напрямую в компонент векторного хранилища, убедитесь, что длина запроса не превышает ограничения модели эмбедингов. В производственной среде может потребоваться добавить проверки или шаги предобработки — например, дополнительные компоненты для подготовки ввода или ограничение длины запроса на уровне кода приложения.

5. В компоненте **Языковая модель** введите ваш API-ключ OpenAI или выберите другого провайдера и модель для части чата в потоке.

6. Запустите подпоток **Загрузка данных**, чтобы заполнить векторное хранилище. В компоненте **Чтение файла** выберите один или несколько файлов, затем нажмите **Run component** на компоненте векторного хранилища в подпотоке **Загрузка данных**. Подпоток загружает файлы с вашего компьютера, разбивает их на фрагменты, генерирует векторные представления и сохраняет их в базе данных. Подпоток **Загрузка данных** отделён от **Поискового модуля**, поскольку загрузка данных не требуется при каждом обращении к чату. Запускайте его по мере необходимости для предварительной загрузки или обновления данных в векторном хранилище. Если хранилище уже содержит нужные данные, запускать подпоток не нужно.
7. Откройте **Playground** и начните чат, чтобы запустить подпоток **Поисковый модуль**. Подпоток генерирует векторное представление из пользовательского ввода, выполняет поиск по сходству в хранилище, передаёт результаты в качестве дополнительного контекста для языковой модели и генерирует ответ на естественном языке. Компонент **Парсер** извлекает строки text из объекта Data результатов поиска и передаёт их компоненту **Шаблон промпта** в формате Message, после чего содержимое шаблона компилируется в инструкцию для языковой модели. Для просмотра необработанных результатов поиска нажмите **Проверить вывод** на компоненте векторного хранилища после выполнения подпотока.

Пример: гибридный поиск

Компонент **Astra DB** поддерживает гибридный поиск Data API. Гибридный поиск объединяет векторный поиск по сходству и лексический поиск, сравнивает их результаты и возвращает наиболее релевантные документы.

Чтобы использовать гибридный поиск через компонент **Astra DB**:

1. С помощью Data API создайте коллекцию с поддержкой гибридного поиска, если у вас её ещё нет. Создание коллекции через Data API даёт больший контроль над её настройками.
2. Создайте поток на основе шаблона **Гибридный поиск RAG** — он включает компонент **Astra DB**, предварительно настроенный для гибридного поиска. После загрузки шаблона проверьте наличие предупреждений **Доступно обновление** на компонентах. При необходимости выполните обновление и переподключите компоненты перед продолжением.
3. В компонентах **Языковая модель** добавьте ваш API-ключ OpenAI. Для использования другого провайдера или модели см. компоненты языковых моделей.
4. Удалите компонент **Языковая модель**, подключённый к компоненту **Структурированный вывод** через порт **Входное сообщение**, и подключите к этому порту компонент **Ввод в чат**.
5. Настройте компонент векторного хранилища **Astra DB**:
 1. Введите токен приложения Astra DB.
 2. В поле **Database** выберите базу данных.
 3. В поле **Коллекция** выберите коллекцию с включённым гибридным поиском.После выбора коллекции, поддерживающей гибридный поиск, остальные параметры обновятся автоматически.
6. Подключите вывод **Обработанный текст** первого компонента **Парсер** к входу **Лексические термины** компонента **Astra DB**. Этот вход появляется только после подключения коллекции с поддержкой гибридного поиска и переранжирования.

7. Обновите шаблон компонента **Структурированный вывод**:

1. Нажмите на компонент **Структурированный вывод**, чтобы открыть панель инспекции компонента.
2. Найдите строку **Инструкции по форматированию**, нажмите **Развернуть** и замените содержимое следующим текстом:
You are a database query planner that takes a user's requests, and then converts to a search against the subject matter in question.
You should convert the query into:
1. A list of keywords to use against a Lucene text analyzer index, no more than 4. Strictly unigrams.
2. A question to use as the basis for a QA embedding engine.
Avoid common keywords associated with the user's subject matter.
3. Нажмите **Завершить редактирование**, затем **Заккрыть**, чтобы сохранить изменения.

8. Откройте **Playground** и введите вопрос на естественном языке о вашей базе данных.

В этом примере ввод отправляется одновременно в компонент **Astra DB** и компонент **Структурированный вывод**:

- Ввод, поступающий напрямую в порт **Поисковый запрос** компонента **Astra DB**, используется как строка для поиска по сходству. Векторное представление генерируется с помощью интеграции vectorize коллекции Astra DB.
- Ввод, поступающий в компонент **Структурированный вывод**, обрабатывается цепочкой компонентов **Структурированный вывод** → **Языковая модель** → **Парсер** для извлечения разделённых пробелами ключевых слов для лексической части гибридного поиска.

Полный запрос гибридного поиска выполняется к базе данных с помощью команды `find_and_rerank` Data API. Ответ API выводится как DataFrame, который преобразуется в текстовое сообщение (Message) следующим компонентом **Парсер**, после чего компонент **Вывод чата** отображает ответ в **Playground**.

9. По желанию: выйдите из **Playground** и нажмите **Проверить вывод** на каждом компоненте, чтобы отследить, как формировались лексические ключевые слова, и просмотреть необработанный ответ Data API. Это полезно при отладке, когда компонент получает неожиданные данные.

- Компонент **Структурированный вывод**: вывод — объект Data, полученный применением схемы к ответу языковой модели. Пример на основе приведённых инструкций по извлечению ключевых слов:
1. Keywords: features, data, attributes, characteristics
2. Question: What characteristics can be identified in my data?
- Компонент **Парсер**: вывод — строка ключевых слов, извлечённых из структурированного вывода Data и используемых в качестве лексических терминов для гибридного поиска.
- Компонент **Astra DB**: вывод — DataFrame, содержащий результаты гибридного поиска, возвращённые Data API.

Вывод Astra DB

При использовании компонента векторного хранилища для запроса базы данных он генерирует результаты поиска, которые можно передать следующим компонентам в потоке в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно задать рядом с портом вывода компонента в визуальном редакторе.

Подключение к порту векторного хранилища

Компонент **Astra DB** имеет дополнительный вывод **Подключение к векторному хранилищу**. Этот вывод может подключаться только к входному порту VectorStore и предназначен для использования с компонентами Graph RAG.

Единственным актуальным компонентом, поддерживающим этот вход, является компонент **Graph RAG**. Рекомендуется использовать компонент **Astra DB Граф**, который сочетает подключение к векторному хранилищу и функциональность Graph RAG.

Astra DB CQL

Компонент **Astra DB CQL** позволяет агентам выполнять запросы к таблицам CQL в Astra DB.

Выходные данные — список объектов Data, содержащих результаты запросов из таблицы Astra DB CQL. Каждый объект Data содержит поля документа, указанные в параметрах проекции. Количество результатов ограничено параметром number_of_results.

Параметры Astra DB CQL

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
Название инструмента	Строка	Имя для ссылки на инструмент в промпте агента.
Описание инструмента	Строка	Краткое описание инструмента, помогающее модели определить, когда его использовать.
Пространство ключей	Строка	Имя keyspace.
Имя таблицы	Строка	Имя таблицы Astra DB CQL для запроса.
Token	SecretString	Токен аутентификации для Astra DB.
Конечная точка API	Строка	Конечная точка API Astra DB.
Projection Fields	Строка	Атрибуты для возврата, разделённые запятыми. По умолчанию: *.
Partition Keys	Словарь	Обязательные параметры, которые модель должна заполнить для выполнения запроса.
Clustering Keys	Словарь	Необязательные параметры для уточнения запроса. Обязательные параметры помечаются восклицательным знаком, например !customer_id.

Название	Тип	Описание
Статические фильтры	Словарь	Пары «атрибут–значение» для фильтрации результатов запроса.
Лимит	Строка	Количество возвращаемых записей.

Graph RAG

Компонент **Graph RAG** использует экземпляр `GraphRetriever` для обхода графа, что позволяет выполнять поиск документов на основе графов в векторном хранилище `Astra DB`. Дополнительную информацию см. в документации `DataStax Graph RAG`.

Параметры Graph RAG

Просмотрите параметры компонента векторного хранилища, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Controls**, пока не заданы необходимые зависимости.

Название	Отображаемое название	Описание
<code>embedding_model</code>	Модель эмбедингов	Модель эмбедингов для использования. Не требуется, если подключённое векторное хранилище имеет интеграцию <code>vectorize</code> .
<code>vector_store</code>	Подключение к векторному хранилищу	Экземпляр <code>AstraDbVectorStore</code> , полученный из выхода Подключение к векторному хранилищу компонента Astra DB .
<code>edge_definition</code>	Определение ребра	Определение ребра для обхода графа.
<code>strategy</code>	Стратегия обхода	Стратегия обхода графа. Параметры динамически загружаются из доступных стратегий.
<code>search_query</code>	Поисковый запрос	Запрос для поиска в векторном хранилище.
<code>graphrag_strategy_kwarg</code>	Параметры стратегии	Необязательный словарь дополнительных параметров для стратегии извлечения.

Название	Отображаемое название	Описание
search_results	Результаты поиска или DataFrame	Результаты поиска документов на основе графа в виде списка объектов Data или табличного DataFrame. Желаемый тип вывода задаётся рядом с портом вывода компонента.

Гиперконвергентная база данных (HCD)

Компонент **Гиперконвергентная база данных (HCD)** использует сервер Data API вашего кластера для чтения и записи в векторное хранилище HCD. Поскольку базовые операции выполняются через Data API (изначально созданный для Astra DB), компонент использует экземпляр AstraDBVectorStore.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ построен на основе LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых функций чтения и записи. Эти экземпляры зависят от провайдера и настраиваются согласно параметрам компонента, таким как строка подключения, имя индекса и схема.

В коде компонента экземпляр обычно создаётся как vector_store, однако некоторые компоненты используют другое имя, например имя провайдера.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от провайдера они могут иметь значения по умолчанию или изменяться через переменные окружения, если те поддерживаются в модуле «Генеративный ИИ». Подробнее см. в справочнике API LangChain и документации провайдера.

При использовании компонента векторного хранилища для запроса базы данных он генерирует результаты поиска, которые можно передать следующим компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат задаётся рядом с портом вывода компонента в визуальном редакторе.

Дополнительную информацию об HCD см. в разделах Get started with HCD 1.2 и Get started with the Data API in HCD 1.2.

Параметры HCD

Просмотрите параметры компонента векторного хранилища, чтобы узнать больше о принимаемых входных данных, поддерживаемых функциях и способах настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после установки других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Controls**, пока не заданы необходимые зависимости.

Название	Отображаемое название	Описание
collection_name	Название коллекции	Имя коллекции векторного хранилища в HCD. При операции записи, если коллекция не

Название	Отображаемое название	Описание
		существует, создаётся новая. Обязательно.
username	Имя пользователя HCD	Имя пользователя для аутентификации в HCD. По умолчанию: hcd-superuser. Обязательно.
password	Пароль HCD	Пароль для аутентификации в HCD. Обязательно.
api_endpoint	Конечная точка API HCD	Конечная точка Data API вашего развёртывания HCD в формате <code>http[s]://CLUSTER_HOST:GATEWAY_PORT</code> , где <code>CLUSTER_HOST</code> — IP-адрес любого узла кластера, а <code>GATEWAY_PORT</code> — порт сервиса API-шлюза. Например: <code>http://192.0.2.250:8181</code> . Обязательно.
ingest_data	Загружаемые данные	Записи для загрузки в векторное хранилище. Применяется только для операций записи.
search_input	Поисковый запрос	Строка запроса для поиска по сходству. Применяется только к операциям чтения.
namespace	Пространство имён	Пространство имён в HCD, содержащее коллекцию из <code>collection_name</code> . По умолчанию: <code>default_namespace</code> .
ca_certificate	Сертификат УЦ	Необязательный сертификат CA для TLS-соединений с HCD.
metric	Метрика	Метрика для расчётов поиска по сходству: <code>cosine</code> , <code>dot_product</code> или <code>euclidean</code> . Это настройка коллекции. При обращении к существующей коллекции

Название	Отображаемое название	Описание
		оставьте пустым, чтобы использовать её метрику. При создании новой коллекции через операцию записи укажите желаемую метрику.
batch_size	Размер батча	Необязательное количество записей для обработки в одном пакете.
bulk_insert_batch_concurrency	Параллелизм пакетной вставки	Необязательный уровень параллелизма для массовых операций записи.
bulk_insert_overwrite_concurrency	Параллелизм вставки с перезаписью	Необязательный уровень параллелизма для массовых операций записи с upsert.
bulk_delete_concurrency	Параллелизм массового удаления	Необязательный уровень параллелизма для массовых операций удаления.
setup_mode	Режим настройки	Режим конфигурации для векторного хранилища: Sync (по умолчанию), Async или Off.
pre_delete_collection	Предварительное удаление коллекции	Удалять ли коллекцию перед созданием новой. По умолчанию: отключено (false).
metadata_indexing_include	Включить индексирование метаданных	Список полей метаданных для индексации при избирательной индексации — только при создании коллекции. Не применяется к существующим коллекциям. Для коллекции можно задать только один параметр *_indexing*. Если ни один не задан, индексируются все поля.
metadata_indexing_exclude	Исключить индексирования метаданных	из Список полей метаданных, исключаемых из индексации при избирательной индексации

Название	Отображаемое название	Описание
		— только при создании коллекции. Не применяется к существующим коллекциям. Для коллекции можно задать только один параметр <code>*_indexing_*</code> . Если ни один не задан, индексируются все поля.
collection_indexing_policy	Политика индексирования коллекции	Словарь для определения политики индексации — только при создании коллекции. Используйте, если требуется индексация подполей или сложная схема, несовместимая со списком. Для коллекции можно задать только один параметр <code>*_indexing_*</code> .
embedding	Эмбединги	Модель эмбедингов, подключаемая через компонент Модель эмбедингов . Компонент не поддерживает дополнительные заголовки аутентификации <code>vectorize</code> , поэтому интеграция <code>vectorize</code> недоступна даже при её наличии в коллекции HCD.
number_of_results	Количество результатов	Количество возвращаемых результатов поиска. По умолчанию: 4. Применяется только к операциям чтения.
search_type	Тип поиска	Тип поиска: Similarity (по умолчанию), Similarity with score threshold или MMR (Max Marginal Relevance). Применяется только к операциям чтения.
search_score_threshold	Порог оценки поиска	Минимальный порог сходства для результатов при типе Similarity with

Название	Отображаемое название	Описание
		score threshold. По умолчанию: 0.
search_filter	Фильтр метаданных поиска	Необязательный словарь фильтров метаданных, применяемый дополнительно к векторному поиску.

Другие компоненты DataStax

Следующие компоненты также включены в пакет **DataStax**.

Astra DB Память чата

Компонент **Astra DB Память чата** извлекает и сохраняет сообщения чата, используя базу данных Astra DB.

Память чата передаётся между компонентами хранения памяти в виде типа данных Memory. Компонент создаёт экземпляр AstraDBChatMessageHistory — класса истории сообщений чата LangChain, использующего Astra DB для хранения.

Компонент **Astra DB Память чата** не рекомендуется для большинства сценариев хранения памяти, поскольку память обычно представляет собой большие JSON-объекты или строки, нередко превышающие максимальный размер документа, поддерживаемый Astra DB.

Компонент **Агент** в модуле «Генеративный ИИ» включает встроенную память чата, включённую по умолчанию. Агентные потоки не требуют внешней базы данных для хранения памяти чата. Подробнее см. в разделе Варианты управления памятью. Подробнее об использовании внешней памяти чата в потоках см. в разделе компонент **История сообщений**.

Параметры Astra DB Память чата

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
collection_name	String	Имя коллекции Astra DB для хранения сообщений. Обязательно.
token	SecretString	Токен аутентификации для доступа к Astra DB. Обязательно.
api_endpoint	SecretString	URL конечной точки API сервиса Astra DB. Обязательно.
namespace	String	Необязательное пространство имён в Astra DB для коллекции.
session_id	MessageText	Уникальный идентификатор сеанса чата. При отсутствии используется текущий ID сеанса.

Устаревшие компоненты DataStax

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, однако рекомендуется заменить их поддерживаемыми компонентами как можно скорее. Рекомендуемые замены указаны в баннере **Устаревшее** на компонентах в ваших потоках, а также в примечаниях к выпуску и документации Генеративный ИИ.

Если вы не уверены, как заменить устаревший компонент, выполните поиск по провайдеру, сервису или названию компонента. Компонент мог быть заменён полностью новым, похожим или новой версией того же компонента в другой категории.

Если очевидной замены нет, рассмотрите возможность адаптации другого компонента под ваш сценарий. Например, многие **Основные компоненты** предоставляют универсальную функциональность, поддерживающую нескольких провайдеров, — в частности компонент **API-запрос**.

Если ни один из вариантов не подходит, используйте код устаревшего компонента для создания пользовательского компонента или начните обсуждение.

Чтобы не поощрять использование устаревших компонентов в новых потоках, они скрыты по умолчанию. В визуальном редакторе нажмите **Настройки компонентов**, чтобы переключить фильтр **Устаревшее**.

Следующие компоненты DataStax находятся в статусе устаревших:

Astra DB Инструмент

Замените компонент **Astra DB Инструмент** на компонент **Astra DB**.

Компонент **Astra DB Инструмент** позволяет выполнять поиск в коллекциях Astra DB, включая гибридный поиск, векторный поиск и поиск по фильтрам. Для специализированных поисков требуется предварительная настройка коллекции.

Выводит список объектов Data, содержащих результаты запросов из Astra DB. Каждый объект Data содержит поля документа, указанные атрибутами проекции. Количество результатов ограничено параметром `number_of_results` и верхним пределом Data API Astra DB в зависимости от типа поиска.

Компонент можно использовать как самостоятельный шаг в потоке или подключить как инструмент для агента, позволяя агенту запрашивать данные из коллекций Astra DB по мере необходимости.

Значения **Название коллекции**, **Токен приложения Astra DB** и **Конечная точка API Astra DB** находятся в вашей установке Astra DB. Подробнее см. в документации Astra DB Serverless.

Название	Тип	Описание
Название инструмента	String	Имя для ссылки на инструмент в промпте агента.
Описание инструмента	String	Краткое описание инструмента, помогающее модели решить, когда его использовать.
Название пространства ключей	String	Имя keyspace в Astra DB. По умолчанию: <code>default_keyspace</code> .

Название	Тип	Описание
Название коллекции	String	Имя коллекции Astra DB для запроса.
Token	SecretString	Токен аутентификации для доступа к Astra DB.
Конечная точка API	String	Конечная точка API Astra DB.
Projection Fields	String	Список атрибутов, разделённых запятыми, возвращаемых из совпадающих документов. По умолчанию: * — все атрибуты, кроме зарезервированных полей таких как \$vector.
Поисковые параметры инструмента	Dict	Фильтры find Data API Astra DB, передаваемые агенту как инструменты. Агент может использовать их в поиске по своему усмотрению.
Статические фильтры	Dict	Пары «атрибут–значение» для фильтрации результатов. Эквивалент фильтров find Data API Astra DB. Статические фильтры включаются в каждый запрос. Используйте без семантического поиска для обычного поиска по фильтрам.
Количество результатов	Int	Максимальное количество возвращаемых документов.
Семантический поиск	Boolean	Выполнять ли поиск по сходству, генерируя векторное представление из ввода чата. По умолчанию: false. При true необходимо подключить компонент модели эмбедингов или включить vectorize в коллекции.

Название	Тип	Описание
Использовать Astra DB vectorize	Boolean	Использовать ли функцию vectorize Astra DB для генерации эмбедингов при семантическом поиске. По умолчанию: false. При true vectorize должна быть предварительно включена в коллекции.
Модель эмбедингов	Эмбединги	Порт для подключения компонента модели эмбедингов для генерации вектора из входного текста при семантическом поиске. Убедитесь, что используется модель с размерностью, совпадающей с эмбедингами в коллекции.
Инструкция для семантического поиска	String	Запрос, используемый для поиска по сходству. По умолчанию: "Find documents similar to the query."

Astra DB Граф

Замените компонент **Astra DB Граф** на компонент **Graph RAG**.

Компонент **Astra DB Граф** использует AstraDBGraphVectorStore — экземпляр графового векторного хранилища LangChain — для обхода графа и получения документов на его основе в коллекции Astra DB. Компонент также поддерживает запись в векторное хранилище.

Подробнее см. в разделе Создание системы Graph RAG с LangChain и GraphRetriever.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры можно изменить через панель инспекции компонента. Некоторые параметры являются условными и появляются только после задания зависимых значений.

Для информации о допустимых значениях см. документацию Astra DB Serverless или код компонента.

Название	Отображаемое название	Описание
token	Токен приложения Astra DB	Токен приложения Astra DB с правом доступа к векторной базе данных. После проверки соединения дополнительные поля заполняются существующими базами данных и коллекциями. Для создания базы данных через

Название	Отображаемое название	Описание
		компонент токена должен иметь права администратора организации.
api_endpoint	Конечная точка API	Конечная точка API вашей базы данных.
keyspace	Пространство ключей	Keyspace в базе данных, содержащий коллекцию из collection_name. По умолчанию: default_keyspace.
collection_name	Коллекция	Имя коллекции для использования в потоке. При операции записи, если коллекция не существует, создаётся новая.
metadata_incoming_links_key	Ключ метаданных входящих ссылок	Ключ метаданных для входящих ссылок в векторном хранилище.
ingest_data	Загружаемые данные	Записи для загрузки в векторное хранилище. Применяется только к операциям записи.
search_input	Поисковый запрос	Строка запроса для поиска по сходству. Применяется только к операциям чтения.
cache_vector_store	Кэшировать векторное хранилище	Кэшировать векторное хранилище в памяти Генеративный ИИ для ускорения чтения. По умолчанию: включено (true).
embedding_model	Модель эмбедингов	Подключите компонент модели эмбедингов для генерации эмбедингов. Не подключайте, если у коллекции есть интеграция vectorize.
metric	Метрика	Метрика для расчётов поиска по сходству: cosine (по умолчанию), dot_product или euclidean. Это настройка коллекции.

Название	Отображаемое название	Описание
batch_size	Размер батча	Необязательное количество записей для обработки в одном пакете.
bulk_insert_batch_concurrency	Параллелизм пакетной вставки	Необязательный уровень параллелизма для массовых операций записи.
bulk_insert_overwrite_concurrency	Параллелизм вставки с перезаписью	Необязательный уровень параллелизма для массовых операций записи с upsert.
bulk_delete_concurrency	Параллелизм массового удаления	Необязательный уровень параллелизма для массовых операций удаления.
setup_mode	Режим настройки	Режим конфигурации векторного хранилища: Sync (по умолчанию) или Off.
pre_delete_collection	Предварительное удаление коллекции	Удалять ли коллекцию перед созданием новой. По умолчанию: отключено (false).
metadata_indexing_include	Включить индексирование метаданных	Список полей метаданных для индексации при избирательной индексации — только при создании коллекции. Для коллекции можно задать только один параметр *_indexing_*. Если ни один не задан, индексируются все поля.
metadata_indexing_exclude	Исключить индексирования метаданных	из Список полей метаданных, исключаемых из индексации при избирательной индексации — только при создании коллекции. Для коллекции можно задать только один параметр *_indexing_*. Если ни один не задан, индексируются все поля.
collection_indexing_policy	Политика индексирования коллекции	Словарь для определения политики

Название	Отображаемое название	Описание
		индексации — только при создании коллекции. Используйте, если требуется индексация подполей или сложная схема. Для коллекции можно задать только один параметр <code>*_indexing_*</code> .
<code>number_of_results</code>	Количество результатов	Количество возвращаемых результатов поиска. По умолчанию: 4. Применяется только к операциям чтения.
<code>search_type</code>	Тип поиска	Тип поиска: Similarity, Similarity with score threshold, MMR (Max Marginal Relevance), Graph Traversal или MMR Graph Traversal (по умолчанию). Применяется только к операциям чтения.
<code>search_score_threshold</code>	Порог оценки поиска	Минимальный порог сходства при типе Similarity with score threshold. По умолчанию: 0.
<code>search_filter</code>	Фильтр метаданных поиска	Необязательный словарь фильтров метаданных, применяемый дополнительно к векторному поиску.

Компоненты Assistants API

Следующие компоненты DataStax использовались для создания и управления функциями Assistants API в потоке:

- **Astra Помощник агента**
- **Создать помощника**
- **Создать тему помощника**
- **Получить имя помощника**
- **Список помощников**
- **Запустить помощника**

Эти компоненты устарели. Замените их нативными агентными компонентами Генеративный ИИ.

Компоненты переменных окружения

Следующие компоненты DataStax использовались для загрузки и получения переменных окружения в потоке:

- **Dotenv**: загружает переменные окружения из файла `.env`.
- **Получить переменную среды**: получает значение переменной окружения.

Эти компоненты устарели. Используйте встроенную поддержку переменных окружения Генеративный ИИ или глобальные переменные.

Astra Vectorize

Компонент объявлен устаревшим в версии Генеративный ИИ 1.1.2. Замените его компонентом **Astra DB**.

Компонент **Astra DB Vectorize** использовался для генерации эмбеддингов с помощью функции `vectorize` Astra DB совместно с компонентом **Astra DB**.

Функциональность `vectorize` теперь встроена непосредственно в компонент **Astra DB** — отдельный компонент для генерации эмбеддингов больше не требуется.

26.24 DeepSeek

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **DeepSeek**.

Дополнительную информацию о функциях и возможностях DeepSeek, используемых компонентами пакета, см. в документации DeepSeek.

Генерация текста DeepSeek

Компонент **DeepSeek** генерирует текст с использованием языковых моделей DeepSeek.

Он может возвращать либо **Ответ модели** (`Message`), либо **Языковую модель** (`LanguageModel`).

Используйте вывод **Языковая модель**, когда хотите применить модель DeepSeek в качестве LLM для другого компонента, управляемого языковой моделью, — например, компонента **Агент** или **Умное преобразование**.

Дополнительную информацию см. в разделе Компоненты языковых моделей.

Параметры генерации текста DeepSeek

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Изменить все параметры компонента можно через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
<code>max_tokens</code>	Целое число	Максимальное количество токенов для генерации. Установите 0 для снятия ограничения. Диапазон: 0–128000.
<code>model_kwargs</code>	Словарь	Дополнительные именованные аргументы для модели.
<code>json_mode</code>	Булево	Если <code>true</code> , выводит ответ в формате JSON вне зависимости от передачи схемы.

Название	Тип	Описание
model_name	Строка	Модель DeepSeek для использования. По умолчанию: deepseek-chat.
api_base	Строка	Базовый URL для API-запросов. По умолчанию: https://api.deepseek.com.
api_key	SecretString	API-ключ DeepSeek для аутентификации.
temperature	Число с плавающей точкой	Управляет случайностью ответов. Диапазон: 0.0–2.0. По умолчанию: 1.0.
seed	Целое число	Начальное значение генератора случайных чисел. Одинаковое значение seed обеспечивает более воспроизводимые результаты; разные значения дают более случайные результаты.

26.25 Docling

Пакеты содержат пользовательские компоненты, поддерживающие интеграцию с конкретными сторонними сервисами в модуле «Генеративный ИИ».

Генеративный ИИ интегрируется с Docling через набор компонентов для парсинга и разбиения документов.

Использование компонентов Docling в потоке

Чтобы узнать больше об извлечении контента с помощью Docling, смотрите видеоруководство Docling + Генеративный ИИ: обработка документов для AI-рабочих процессов. В этом примере показано, как использовать компоненты Docling для разбиения PDF в потоке:

1. Подключите компонент **Docling** и компонент **Экспорт DoclingDocument** к компоненту **Разделить текст**.
Компонент **Docling** загружает документ, а компонент **Экспорт DoclingDocument** преобразует DoclingDocument в выбранный формат. В примере документ преобразуется в Markdown с заполнителями вместо изображений. Компонент **Разделить текст** разбивает Markdown на фрагменты для векторной базы данных, которая заполняется в следующей части потока.
2. Подключите компонент векторного хранилища **Chroma DB** к выходу **Чанки** компонента **Разделить текст**.
3. Подключите компонент модели эмбедингов к порту **Эмбединги** компонента **Chroma DB**, а также компонент **Вывод чата** для просмотра извлечённого DataFrame.
4. В компоненте модели эмбедингов выберите предпочтительную модель, укажите учётные данные и при необходимости настройте остальные параметры.

5. Добавьте файл в компонент **Docling**.
6. Нажмите **Playground**, чтобы запустить поток.

Фрагментированный документ загрузится в виде векторов в вашу векторную базу данных.

Компоненты Docling

В следующих разделах описываются назначение и параметры конфигурации каждого компонента набора **Docling**.

Локальная модель Docling

Компонент **Docling** принимает документы и обрабатывает их с помощью локальной модели Docling.

Выходные данные — files: обработанные файлы с данными DoclingDocument.

Дополнительную информацию см. в репозитории проекта Docling IBM models.

Параметры Docling

Название	Тип	Описание
files	Строка	Файлы для обработки.
pipeline	Строка	Конвейер Docling для использования (standard, vlm).
ocr_engine	Строка	Движок OCR для использования (easyocr, tesseract, rapidocr, ocrm).

Docling Serve

Компонент **Docling Serve** принимает документы и обрабатывает их через API-сервис Docling вместо локальной модели.

Выходные данные — files: обработанные файлы с данными DoclingDocument.

Дополнительную информацию см. в репозитории проекта Docling Serve.

Параметры Docling Serve

Название	Тип	Описание
files	Строка	Файлы для обработки.
api_url	Строка	URL экземпляра Docling Serve.
max_concurrency	Целое число	Максимальное количество одновременных запросов к серверу.
max_poll_timeout	Число с плавающей точкой	Максимальное время ожидания завершения конвертации документа.
api_headers	Словарь	Необязательный словарь дополнительных заголовков для подключения к Docling Serve.
docling_serve_opts	Словарь	Необязательный словарь дополнительных параметров Docling Serve.

Фрагментирование DoclingDocument

Компонент **Фрагментирование DoclingDocument** разбивает объекты DoclingDocument на фрагменты.

Выходные данные — фрагментированные документы в виде DataFrame.

Дополнительную информацию см. в репозитории проекта Docling core.

Параметры фрагментирования DoclingDocument

Название	Тип	Описание
data_inputs	Data/DataFrame	Данные с документами, которые необходимо разбить на фрагменты.
chunker	Строка	Чанкер для использования (HybridChunker, HierarchicalChunker).
provider	Строка	Поставщик токенизатора (Hugging Face, OpenAI).
hf_model_name	Строка	Название модели токенизатора для HybridChunker при выборе Hugging Face.
openai_model_name	Строка	Название модели токенизатора для HybridChunker при выборе OpenAI.
max_tokens	Целое число	Максимальное количество токенов для HybridChunker.
doc_key	Строка	Ключ столбца DoclingDocument.

Экспорт DoclingDocument

Компонент **Экспорт DoclingDocument** экспортирует DoclingDocument в формат Markdown, HTML и другие.

Экспортированные данные могут выводиться как Data или как DataFrame.

Дополнительную информацию см. в репозитории проекта Docling core.

Параметры экспорта DoclingDocument

Название	Тип	Описание
data_inputs	Data/DataFrame	Данные с документами, которые необходимо экспортировать.
export_format	Строка	Формат экспорта (Markdown, HTML, Plaintext, DocTags).
image_mode	Строка	Способ экспорта изображений (placeholder, embedded).

Название	Тип	Описание
md_image_placeholder	Строка	Заполнитель изображения для экспорта в Markdown.
md_page_break_placeholder	Строка	Заполнитель разрыва страницы между страницами в Markdown-выводе.
doc_key	Строка	Ключ столбца DoclingDocument.

26.26 DuckDuckGo

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **DuckDuckGo**.

DuckDuckGo Search

Этот компонент выполняет веб-поиск с использованием поисковой системы DuckDuckGo с возможностью ограничения количества результатов.

Компонент возвращает список результатов поиска в виде DataFrame с ключом text, содержащим результаты поиска в виде одной строки.

Параметры DuckDuckGo Search

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
input_value	String	Входной параметр. Поисковый запрос для выполнения через DuckDuckGo.
max_results	Integer	Входной параметр. Максимальное количество возвращаемых результатов поиска. По умолчанию: 5.
max_snippet_length	Integer	Входной параметр. Максимальная длина каждого фрагмента результата. По умолчанию: 100.

26.27 Elastic

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Elastic**.

Elasticsearch

Компонент **Elasticsearch** читает и записывает данные в экземпляр Elasticsearch, используя ElasticsearchStore.

Об экземплярах векторного хранилища

Поскольку Генеративный ИИ построен на основе LangChain, компоненты векторных хранилищ используют экземпляр векторного хранилища LangChain для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента, такими как строка подключения, имя индекса и схема.

В коде компонента такой экземпляр, как правило, создаётся как `vector_store`, однако некоторые компоненты векторных хранилищ используют другое имя — например, имя поставщика.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от поставщика эти параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Сведения о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

При использовании компонента векторного хранилища для запроса к векторной базе данных он возвращает результаты поиска, которые можно передать последующим компонентам потока в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба формата, выбрать нужный можно рядом с портом выхода компонента векторного хранилища в визуальном редакторе.

Параметры Elasticsearch

Просмотрите параметры компонента векторного хранилища, чтобы узнать о принимаемых входных данных, поддерживаемых функциях и способах настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и отображаются только после установки других параметров или выбора определённых значений. Условные параметры могут отсутствовать на панели **Controls** до тех пор, пока не заданы необходимые зависимости.

Сведения о допустимых значениях и функциональности см. в документации Elasticsearch или в коде компонента.

Название	Тип	Описание
elasticsearch_url	String	Входной параметр. URL сервера Elasticsearch.
cloud_id	String	Входной параметр. Cloud ID Elasticsearch.
index_name	String	Входной параметр. Имя индекса Elasticsearch.
ingest_data	Data	Входной параметр. Записи для загрузки в векторное хранилище.
search_query	String	Входной параметр. Поисковый запрос для поиска по сходству.

Название	Тип	Описание
cache_vector_store	Boolean	Входной параметр. Если true, компонент кэширует векторное хранилище в памяти для ускорения операций чтения. По умолчанию: true.
username	String	Входной параметр. Имя пользователя для аутентификации в Elasticsearch. Обязательно для всех локальных развёртываний. Обязательно для облачных развёртываний, если api_key не задан.
password	SecretString	Входной параметр. Пароль для аутентификации в Elasticsearch. Обязательно для всех локальных развёртываний. Обязательно для облачных развёртываний, если api_key не задан.
embedding	Embeddings	Входной параметр. Модель эмбедингов для использования.
search_type	String	Входной параметр. Тип поиска. Варианты: similarity (по умолчанию) или mmr.
number_of_results	Integer	Входной параметр. Количество результатов поиска. По умолчанию: 4.
search_score_threshold	Float	Входной параметр. Минимальный порог оценки сходства для результатов поиска. По умолчанию: 0.
api_key	SecretString	Входной параметр. API-ключ для аутентификации в Elastic Cloud. Если указан, username и password не требуются.
verify_certs	Boolean	Входной параметр. Проверять ли SSL-

Название	Тип	Описание
		сертификаты при подключении к Elasticsearch. По умолчанию: true.

OpenSearch

Компонент **OpenSearch** читает и записывает данные в экземпляры OpenSearch, используя OpenSearchVectorSearch.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр векторного хранилища LangChain для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента, такими как строка подключения, имя индекса и схема.

В коде компонента такой экземпляр, как правило, создаётся как `vector_store`, однако некоторые компоненты векторных хранилищ используют другое имя — например, имя поставщика.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от поставщика эти параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Сведения о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Параметры OpenSearch

Просмотрите параметры компонента векторного хранилища, чтобы узнать о принимаемых входных данных, поддерживаемых функциях и способах настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и отображаются только после установки других параметров или выбора определённых значений. Условные параметры могут отсутствовать на панели **Controls** до тех пор, пока не заданы необходимые зависимости.

Сведения о допустимых значениях и функциональности см. в документации OpenSearch или в коде компонента.

Название	Тип	Описание
<code>opensearch_url</code>	String	Входной параметр. URL кластера OpenSearch, например <code>https://192.168.1.1:9200</code> .
<code>index_name</code>	String	Входной параметр. Имя индекса, в котором хранятся векторы в кластере OpenSearch. По умолчанию: "astra".
<code>ingest_data</code>	Data	Входной параметр. Данные для загрузки в векторное хранилище.

Название	Тип	Описание
search_input	String	Входной параметр. Поисковый запрос. Оставьте пустым, чтобы получить все документы или при использовании гибридного поиска.
cache_vector_store	Boolean	Входной параметр. Если true, компонент кэширует векторное хранилище в памяти для ускорения операций чтения. По умолчанию: true.
embedding	Embeddings	Входной параметр. Подключите компонент модели эмбедингов для генерации эмбедингов из поискового запроса.
search_type	String	Входной параметр. Тип поиска. Варианты: similarity (по умолчанию), similarity_score_threshold, mmr.
number_of_results	Integer	Входной параметр. Количество результатов поиска. По умолчанию: 4.
search_score_threshold	Float	Входной параметр. Минимальный порог оценки сходства для результатов поиска. По умолчанию: 0.
username	String	Входной параметр. Имя пользователя для кластера OpenSearch. По умолчанию: "admin".
password	SecretString	Входной параметр. Пароль для кластера OpenSearch.
use_ssl	Boolean	Входной параметр. Использовать ли SSL. По умолчанию: true.
verify_certs	Boolean	Входной параметр. Проверять ли SSL-сертификаты. По умолчанию: false.

Название	Тип	Описание
hybrid_search_query	String	Входной параметр. Пользовательский гибридный поисковый запрос в формате JSON, комбинирующий векторное сходство и поиск по ключевым словам.

Выходы OpenSearch

При использовании компонента векторного хранилища для запроса к векторной базе данных он возвращает результаты поиска, которые можно передать последующим компонентам потока в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба формата, выбрать нужный можно рядом с портом выхода компонента векторного хранилища в визуальном редакторе.

Подключение к порту векторного хранилища

Компонент **OpenSearch** имеет дополнительный устаревший выход **Подключение к векторному хранилищу**. Этот выход может подключаться только к входному порту VectorStore и предназначен для использования с отдельными компонентами Graph RAG.

Компонент **OpenSearch** не требует отдельного компонента Graph RAG, поскольку экземпляры OpenSearch поддерживают обход графа через встроенную функциональность RAG и плагины.

26.28 Eха

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Eха**.

Eха Search

Этот компонент предоставляет набор инструментов Eха Search для поиска и получения контента с помощью компонента **Агент** или MCP-клиента.

Компонент возвращает исключительно Tools.

Параметры Eха Search

Название	Тип	Описание
API-ключ (metaphor_api_key)	SecretString	Входной параметр. API-ключ для Eха Search.
Автоподсказка (use_autoprompt)	Boolean	Входной параметр. Использовать ли функцию автоподсказки. По умолчанию: true.
Количество результатов поиска (search_num_results)	Integer	Входной параметр. Количество результатов, возвращаемых при поиске. По умолчанию: 5.
Количество похожих результатов (similar_num_results)	Integer	Входной параметр. Количество похожих

Название	Тип	Описание
		результатов, возвращаемых в ответе. По умолчанию: 5.

26.29 FAISS

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **FAISS**.

Векторное хранилище FAISS

Компонент **FAISS** предоставляет доступ к библиотеке Facebook AI Similarity Search (FAISS) через экземпляр векторного хранилища FAISS.

Об экземплярах векторного хранилища

Поскольку Генеративный ИИ основан на LangChain, компоненты векторного хранилища используют экземпляр векторного хранилища LangChain для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента, такими как строка подключения, имя индекса и схема.

В коде компонента такой экземпляр, как правило, создаётся как `vector_store`, однако некоторые компоненты векторного хранилища используют другое имя — например, имя поставщика.

Некоторые классы LangChain не раскрывают все возможные параметры в виде параметров компонента. В зависимости от поставщика эти параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Сведения о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

При использовании компонента векторного хранилища для запроса к векторной базе данных он возвращает результаты поиска, которые можно передать последующим компонентам потока в виде списка объектов `Data` или табличного `DataFrame`. Если поддерживаются оба формата, выбрать нужный можно рядом с портом выхода компонента векторного хранилища в визуальном редакторе.

Параметры векторного хранилища FAISS

Просмотрите параметры компонента, чтобы узнать о принимаемых входных данных, поддерживаемых функциях и способах настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и отображаются только после установки других параметров или выбора определённых значений. Условные параметры могут отсутствовать на панели **Controls** до тех пор, пока не заданы необходимые зависимости.

Сведения о допустимых значениях и функциональности см. в документации FAISS или в коде компонента.

Название	Тип	Описание
<code>index_name</code>	String	Входной параметр. Имя индекса FAISS. По умолчанию: "astra_index".

Название	Тип	Описание
<code>persist_directory</code>	String	Входной параметр. Путь для сохранения индекса FAISS. Путь задаётся относительно рабочего каталога Генеративный ИИ.
<code>search_query</code>	String	Входной параметр. Запрос для поиска в векторном хранилище.
<code>ingest_data</code>	Data	Входной параметр. Список данных для загрузки в векторное хранилище.
<code>allow_dangerous_deserialization</code>	Boolean	Входной параметр. Задайте True, чтобы разрешить загрузку pickle-файлов из ненадёжных источников. По умолчанию: True.
<code>embedding</code>	Embeddings	Входной параметр. Функция эмбедингов, используемая для векторного хранилища.
<code>number_of_results</code>	Integer	Входной параметр. Количество результатов, возвращаемых при поиске. По умолчанию: 4.

26.30 Glean

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Glean**.

Glean Search API

Этот компонент позволяет вызывать Glean Search API. [web:708]

Компонент возвращает список результатов поиска в виде DataFrame.

Параметры Glean Search API

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
<code>glean_api_url</code>	String	Входной параметр. URL API Glean.
<code>glean_access_token</code>	SecretString	Входной параметр. Токен доступа для

Название	Тип	Описание
		аутентификации в API Glean.
query	String	Входной параметр. Текст поискового запроса.
page_size	Integer	Входной параметр. Количество результатов на страницу. По умолчанию: 10.
request_options	Dict	Входной параметр. Дополнительные параметры запроса API.

26.31 Компоненты Google

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Google**.

BigQuery

Генеративный ИИ интегрируется с Google BigQuery через компонент **BigQuery** в пакете **Google**, позволяя выполнять SQL-запросы и получать данные из ваших наборов данных BigQuery.

Использование компонента BigQuery в потоке

Для использования компонента **BigQuery** в потоке потребуется следующее:

- Проект Google Cloud с включённым API BigQuery
- Сервисный аккаунт с ролью **BigQuery Job User**
- Набор данных и таблица BigQuery
- Запущенный сервер Генеративный ИИ

Создание сервисного аккаунта с доступом к BigQuery

1. Выберите и включите ваш проект Google Cloud. Подробнее см. в разделе Создание проекта Google Cloud.
2. Создайте сервисный аккаунт в вашем проекте Google Cloud. Подробнее см. в разделе Создание сервисного аккаунта.
3. Назначьте роль **BigQuery Job User** новому аккаунту. Эта роль позволяет Генеративный ИИ получать доступ к ресурсам BigQuery через сервисный аккаунт. Возможно, также потребуется предоставить доступ к вашему набору данных BigQuery. Подробнее см. в разделе BigQuery access control with IAM.
4. Перейдите к вашему сервисному аккаунту, чтобы создать новый JSON-ключ.
5. Нажмите **Добавить ключ**, затем **Создать новый ключ**.
6. В поле **Тип ключа** выберите **JSON** и нажмите **Создать**. Файл с приватным ключом JSON будет загружен на ваш компьютер. После получения сервисного аккаунта и файла JSON-ключа настройте учётные данные в компоненте **BigQuery**.

Настройка учётных данных в компоненте Генеративный ИИ

Имея настроенный сервисный аккаунт и файл JSON-учётных данных, выполните следующие шаги для аутентификации Генеративный ИИ.

1. Создайте новый поток в модуле «Генеративный ИИ».

2. В разделе **Пакеты** найдите компонент **BigQuery** и добавьте его в поток.
3. В поле **Загрузить сервисный аккаунт в формате JSON** компонента **BigQuery** нажмите **Выбрать файлы**.
4. В панели **Мои файлы** выберите **Кликнуть или перетащить файлы сюда**. Откроется файловый менеджер.
5. Выберите файл JSON вашего сервисного аккаунта и нажмите **Open**.
6. В панели **Мои файлы** выберите ваш файл JSON и нажмите **Выбрать файлы**. Теперь компонент **BigQuery** может выполнять запросы к наборам данных и таблицам, используя файл JSON сервисного аккаунта.

Запрос к набору данных BigQuery

После настройки учётных данных выполните запрос к наборам данных и таблицам BigQuery, чтобы подтвердить соединение.

1. Подключите компоненты **Ввод в чат** и **Вывод чата** к компоненту **BigQuery**.
2. Откройте **Playground** и отправьте корректный SQL-запрос.

В этом примере запрашивается таблица победителей премии «Оскар», хранящаяся в наборе данных BigQuery под названием the_oscar_award:

```
SELECT film, category, year_film
FROM `big-query-astra-project.the_oscar_award.oscar_winners`
WHERE winner = TRUE
LIMIT 10
```

Результат

```
film    category    year_film
The Last Command  ACTOR    1927
7th Heaven    ACTRESS  1927
The Dove;    ART DIRECTION  1927
Sunrise    CINEMATOGRAPHY  1927
Sunrise    CINEMATOGRAPHY  1927
Two Arabian Knights  DIRECTING (Comedy Picture)  1927
7th Heaven    DIRECTING (Dramatic Picture)  1927
Wings    ENGINEERING EFFECTS  1927
Wings    OUTSTANDING PICTURE  1927
Sunrise    UNIQUE AND ARTISTIC PICTURE  1927
```

Успешный ответ подтверждает, что компонент может получить доступ к таблице BigQuery.

Google Generative AI: генерация текста

Этот компонент генерирует текст с использованием моделей Google Generative AI.

Поддерживаемые модели включают серии Gemini 1.5, 2.0, 2.5 и 3.0. Последние модели Gemini 3.0 (gemini-3-pro-preview, gemini-3-flash-preview и gemini-3-pro-image-preview) предлагают расширенные возможности рассуждения и мультимодальности.

Параметры Google Generative AI

Название	Тип	Описание
API-ключ Google	SecretString	Входной параметр. API-ключ Google для использования Google Generative AI.
Модель	String	Входной параметр. Имя модели для использования, например "gemini-2.5-flash" или "gemini-3-pro-preview".

Название	Тип	Описание
Максимальное количество токенов вывода	Integer	Входной параметр. Максимальное количество токенов для генерации.
Температура	Float	Входной параметр. Температура выборки при генерации вывода.
Количество лучших токенов	Integer	Входной параметр. Значение top-K: количество наиболее вероятных токенов, учитываемых при выборке.
Вероятностная масса	Float	Входной параметр. Значение top-P: максимальная суммарная вероятность токенов, учитываемых при выборке.
Количество вариантов	Integer	Входной параметр. Количество вариантов ответа, генерируемых для каждого запроса.
model	LanguageModel	Выходной параметр. Экземпляр ChatGoogleGenerativeAI, настроенный с указанными параметрами.

Google Generative AI: эмбединги

Компонент **Google Generative AI** подключается к сервису эмбедингов Google Generative AI с использованием класса `GoogleGenerativeAIEmbeddings` из пакета `langchain-google-genai`.

Дополнительные сведения об использовании компонентов моделей эмбедингов в потоках см. в разделе [Компоненты моделей эмбедингов](#).

Параметры эмбедингов Google Generative AI

Название	Отображаемое название	Описание
api_key	API-ключ	Входной параметр. Секретный API-ключ для доступа к сервису Google Generative AI. Обязательно.
model_name	Название модели	Входной параметр. Имя модели эмбедингов для использования. По умолчанию: "models/text-embedding-004".
embeddings	Эмбединги	Выходной параметр. Созданный объект

Название	Отображаемое название	Описание
		GoogleGenerativeAIEmbeddings.

API поиска Google

Этот компонент позволяет вызывать Google Search API.

Параметры API поиска Google

Название	Тип	Описание
google_api_key	SecretString	Входной параметр. API-ключ Google для аутентификации.
google_cse_id	SecretString	Входной параметр. Идентификатор поисковой системы Google Custom Search.
input_value	String	Входной параметр. Текст поискового запроса.
k	Integer	Входной параметр. Количество возвращаемых результатов поиска.
results	List[Data]	Выходной параметр. Список результатов поиска.
tool	Tool	Выходной параметр. Инструмент Google Search для использования в LangChain.

Другие компоненты поиска Google

Генеративный ИИ включает несколько компонентов, поддерживающих поиск Google:

- Компонент **Apify Actors**
- Компонент **SearchApi**
- Компонент **Serper API поиска Google**
- Компонент **Веб-поиск**

Google Vertex AI

Сведения о компонентах Vertex AI см. в разделе пакет **Vertex AI**.

Устаревшие компоненты Google

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, однако рекомендуется заменить их поддерживаемыми компонентами как можно скорее. Рекомендуемые замены указаны в баннере **Устаревшее** на соответствующих компонентах в ваших потоках, а также в примечаниях к выпускам и документации Генеративный ИИ, когда это возможно.

Если вы не уверены, как заменить устаревший компонент, воспользуйтесь **Поиском** компонентов по поставщику, сервису или названию. Компонент мог быть объявлен устаревшим в пользу нового компонента, похожего компонента или новой версии того же компонента в другой категории.

Если явной замены нет, рассмотрите возможность адаптации другого компонента под ваш сценарий. Например, многие **Основные компоненты** предоставляют универсальную функциональность, поддерживающую нескольких поставщиков и сценариев использования, — в частности, компонент **API-запрос**.

Если ни один из вариантов не подходит, вы можете использовать код устаревшего компонента для создания собственного пользовательского компонента или открыть обсуждение об устаревшем компоненте.

Чтобы не поощрять использование устаревших компонентов в новых потоках, они скрыты по умолчанию. В визуальном редакторе нажмите **Настройки компонента**, чтобы переключить фильтр **Устаревшее**.

Следующие компоненты Google находятся в статусе устаревших:

Токен Google OAuth

Компонент **Токен Google OAuth** был объявлен устаревшим в модуле «Генеративный ИИ» 1.4.0.

Для подключения потоков к сервисам Google OAuth используйте компоненты Composio.

Загрузчик Gmail

Этот компонент загружает электронные письма из Gmail, используя учётные данные JSON сервисного аккаунта и фильтры по идентификаторам меток.

В качестве альтернативы используйте компоненты Composio для подключения потоков к сервисам Google.

Загрузчик Google Диска

Этот компонент загружает документы из Google Drive, используя учётные данные JSON сервисного аккаунта и фильтры по идентификаторам документов.

Прямой замены нет; рассмотрите использование компонента **API-запрос** для вызова Google Drive API.

Поиск на Google Диске

Этот компонент выполняет поиск файлов в Google Drive, используя учётные данные JSON сервисного аккаунта и различные строки запросов и фильтры.

Прямой замены нет; рассмотрите использование компонента **API-запрос** для вызова Google Drive API.

26.32 Groq

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Groq**.

Дополнительные сведения см. в документации Groq.

Groq: генерация текста

Этот компонент генерирует текст с использованием языковых моделей Groq.

Компонент может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel). Выход **Языковая модель** представляет собой экземпляр ChatGroq, настроенный в соответствии с параметрами компонента.

Используйте выход **Языковая модель**, когда необходимо применять модель Groq в качестве LLM для другого компонента, которым управляет LLM, — например, компонента **Агент** или **Умное преобразование**.

Дополнительные сведения см. в разделе Компоненты языковой модели.

Параметры генерации текста Groq

Название	Тип	Описание
groq_api_key	SecretString	API-ключ Groq.
groq_api_base	String	Базовый URL для запросов к API. По умолчанию: https://api.groq.com .
max_tokens	Integer	Максимальное количество токенов для генерации.
temperature	Float	Управляет случайностью вывода. Диапазон: [0.0, 1.0]. По умолчанию: 0.1.
n	Integer	Количество вариантов ответа, генерируемых для каждого запроса.
model_name	String	Имя модели Groq для использования. Варианты динамически получаются из API Groq после ввода API-ключа и URL. Чтобы обновить список моделей, нажмите Refresh .
tool_mode_enabled	Boolean	Если включено, отображаются только модели, совместимые с инструментами.

26.33 Hugging Face

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Компоненты в пакете **Hugging Face** требуют доступа к API Hugging Face.

Для получения дополнительной информации о функциях и возможностях Hugging Face, используемых компонентами Hugging Face, см. документацию Hugging Face.

Hugging Face генерация текста

Компонент **Hugging Face** генерирует Текст с использованием указанной модели, отправляя запросы к API Hugging Face, который представляет собой размещённый API вывода для моделей, размещённых на Hugging Face. Требуется аутентификация.

Этот компонент может выдавать либо **Ответ Модели** (Message), либо **Языковая модель** (LanguageModel). В частности, вывод **Языковая модель** представляет собой экземпляр ChatHuggingFace, настроенный в соответствии с параметрами компонента.

Используйте вывод **Языковая модель**, когда хотите использовать модель Hugging Face в качестве LLM для другого компонента, управляемого LLM, например, компонента **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. Язык компоненты моделей.

Hugging Face Текст параметры генерации

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Вы можете изменить все параметры компонента через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
model_id	Строка	Входной параметр. Идентификатор модели из Hugging Face Hub. Например, “gpt2”, “facebook/bart-large”.
huggingfacehub_api_token	Секретная строка	Входной параметр. Ваш токен API Hugging Face для аутентификации.
temperature	Вещественное	Входной параметр. Управляет случайностью вывода. Диапазон: [0.0, 1.0]. По умолчанию: 0.7.
max_new_tokens	Целое	Входной параметр. Максимальное количество токенов для генерации. По умолчанию: 512.
top_p	Вещественное	Входной параметр. Параметр ядерной выборки. Диапазон: [0.0, 1.0]. По умолчанию: 0.95.
top_k	Целое	Входной параметр. Параметр выборки top-k. По умолчанию: 50.
model_kwargs	Словарь	Входной параметр. Дополнительные именованные аргументы, передаваемые модели.

Вывод эмбеддингов Hugging Face

Используйте компонент **Вывод эмбеддингов Hugging Face** для создания эмбеддингов с помощью размещённых моделей Hugging Face или ваших собственных локально размещённых моделей.

Компонент генерирует эмбеддинги, используя модели Hugging Face Inference API models. Требуется аутентификация, если не используется локальная модель.

Для получения дополнительной информации об использовании компонентов моделей эмбеддингов в потоках см. Эмбеддинги компоненты модели и Использовать локальную модель эмбеддингов Hugging Face.

Параметры Вывод эмбеддингов Hugging Face

Название	Отображать Название	Информация
api_key	Ключ API	Входной параметр. Ваш Hugging Face API token для доступа к Hugging Face Inference API, если требуется. Локальные модели инференса не требуют API-ключа.
api_url	URL API	Входной параметр. URL Hugging Face Inference API.
model_name	Название модели	Входной параметр. Имя модели, используемой для эмбедингов.

Использовать локальную модель эмбедингов Hugging Face

Чтобы подключить локальную модель Hugging Face к компоненту **Вывод эмбедингов Hugging Face** и использовать её в потоке, выполните следующие шаги:

1. Запустите локальный инференс эмбедингов Hugging Face.
2. Для этого примера создайте поток из шаблона **Векторное хранилище RAG**.
3. Замените два компонента **OpenAI Эмбединги** на компоненты **Вывод эмбедингов Hugging Face**.
Убедитесь, что порты **Модель эмбедингов** от каждого компонента **Эмбединги Inference** подключены к соответствующему компоненту **Astra DB**.
4. Настройте компоненты **Astra DB**, чтобы подключиться к вашей организации Astra, или замените оба компонента **Astra DB** другими компонентами векторного хранилища.
5. Подключите каждый компонент **Вывод эмбедингов Hugging Face** к вашей локальной модели инференса:
 - **Конечная точка вывода:** Укажите URL вашей локальной модели инференса.
 - **Ключ API:** Может быть пустым для локального инференса.
 - **Название модели:** Укажите имя вашей локальной модели инференса, если оно не обнаружено автоматически.
6. Чтобы протестировать поток, нажмите **Playground**, а затем введите Текст, чтобы сгенерировать эмбединг.

26.34 IBM

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Пакет **IBM** предоставляет доступ к моделям IBM watsonx.ai для генерации текста и эмбедингов. Для работы этих компонентов требуется развёртывание IBM watsonx.ai и учётные данные API watsonx.

IBM watsonx.ai: генерация текста

Компонент **IBM watsonx.ai** генерирует текст с использованием поддерживаемых базовых моделей в IBM watsonx.ai. Для использования шлюзовых моделей используйте компонент генерации текста **OpenAI** с совместимой с OpenAI конечной точкой шлюзовой модели.

Компонент **IBM watsonx.ai** можно использовать в любом месте потока, где требуется языковая модель.

Параметры IBM watsonx.ai

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
url	String	Входной параметр. Базовый URL API watsonx для вашего развёртывания и региона.
project_id	String	Входной параметр. Идентификатор проекта watsonx.
api_key	SecretString	Входной параметр. API-ключ watsonx для аутентификации доступа к API watsonx в указанном развёртывании и модели.
model_name	String	Входной параметр. Имя модели watsonx для использования. Параметры динамически получаются из API.
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. По умолчанию: 1000.
stop_sequence	String	Входной параметр. Последовательность, при достижении которой генерация останавливается.
temperature	Float	Входной параметр. Управляет случайностью вывода. По умолчанию: 0.1.
top_p	Float	Входной параметр. Параметр ядерной выборки (nucleus sampling): ограничивает модель токенами с суммарной вероятностью не менее top_p. По умолчанию: 0.9.
frequency_penalty	Float	Входной параметр. Штраф за частоту: положительное значение

Название	Тип	Описание
		снижает вероятность повторения токенов, отрицательное — увеличивает её. По умолчанию: 0.5.
presence_penalty	Float	Входной параметр. Штраф за присутствие: положительное значение повышает вероятность появления новых тем. По умолчанию: 0.3.
seed	Integer	Входной параметр. Начальное значение генератора случайных чисел. По умолчанию: 8.
logprobs	Boolean	Входной параметр. Возвращать ли лог-вероятности токенов вывода. По умолчанию: true.
top_logprobs	Integer	Входной параметр. Количество наиболее вероятных токенов, возвращаемых для каждой позиции. По умолчанию: 3.
logit_bias	String	Входной параметр. JSON-строка с идентификаторами токенов для смещения или подавления.

Выходы IBM watsonx.ai

Компонент **IBM watsonx.ai** может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда необходимо применять модель IBM watsonx.ai в качестве LLM для другого компонента, которым управляет LLM, — например, компонента **Агент** или **Умное преобразование**. Дополнительные сведения см. в разделе **Компоненты языковой модели**.

Выход LanguageModel компонента **IBM watsonx.ai** представляет собой экземпляр ChatWatsonx, настроенный в соответствии с параметрами компонента.

IBM watsonx.ai: эмбединги

Компонент **IBM watsonx.ai** использует поддерживаемые базовые модели в IBM watsonx.ai для генерации эмбедингов.

Выход представляет собой объект Embeddings, сгенерированный с помощью WatsonxEmbeddings.

Дополнительные сведения об использовании компонентов моделей эмбедингов в потоках см. в разделе Компоненты моделей эмбедингов.

Параметры эмбедингов IBM watsonx.ai

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
url	Конечная точка API watsonx	Входной параметр. Базовый URL API watsonx для вашего развёртывания и региона.
project_id	Идентификатор проекта watsonx	Входной параметр. Идентификатор проекта watsonx.
api_key	API-ключ	Входной параметр. API-ключ watsonx для аутентификации доступа к указанному развёртыванию и модели.
model_name	Название модели	Входной параметр. Имя модели эмбедингов для использования. Поддерживает модели эмбедингов по умолчанию и автоматически обновляется после подключения к вашему развёртыванию watsonx.ai.
truncate_input_tokens	Усечение входных токенов	Входной параметр. Максимальное количество токенов для обработки. По умолчанию: 200.
input_text	Включить исходный текст в вывод	Входной параметр. Включать ли исходный текст в вывод. По умолчанию: true.

Модели эмбедингов по умолчанию

По умолчанию компонент **IBM watsonx.ai** поддерживает следующие модели: [web:668][web:670]

- sentence-transformers/all-minilm-l12-v2: 384-мерные эмбединги
- ibm/slate-125m-english-rtrvr-v2: 768-мерные эмбединги
- ibm/slate-30m-english-rtrvr-v2: 768-мерные эмбединги
- intfloat/multilingual-e5-large: 1024-мерные эмбединги

После ввода конечной точки API и учётных данных компонент автоматически получает список доступных моделей из вашего развёртывания watsonx.ai.

26.35 Icosa Computing

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Компоненты **Icosa Computing** требуют доступа к сервисам Icosa Computing. Дополнительные сведения и запрос доступа см. на сайте Icosa Computing.

Комбинаторный рассуждатель

Компонент **Combinatorial Reasoner** запускает конвейер комбинаторного рассуждения (CR) от Icosa на входных данных, чтобы сформировать оптимизированный промпт со встроенными обоснованиями.

Параметры комбинаторного рассуждателя

Название	Тип	Описание
prompt	String	Входной параметр. Промпт для выполнения CR.
openai_api_key	SecretString	Входной параметр. API-ключ OpenAI для аутентификации в OpenAI, используемый для генерации обоснований.
model_name	String	Входной параметр. Модель OpenAI LLM для генерации обоснований.
username	String	Входной параметр. Имя пользователя для аутентификации в API Icosa.
password	SecretString	Входной параметр. Пароль для аутентификации в API Icosa.

Выходы комбинаторного рассуждателя

Компонент **Combinatorial Reasoner** возвращает один из следующих вариантов:

- **Оптимизированный промпт (optimized_prompt)**: объект Message, содержащий оптимизированный промпт со встроенными обоснованиями.
- **Выбранные обоснования (reasons)**: объект Data, содержащий список строк, где каждая строка — это обоснование, выбранное и встроенное в оптимизированный промпт.

Выберите нужный тип рядом с портом выхода компонента.

26.36 LangChain

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **LangChain**.

CSV-агент

Установка `allow_dangerous_code` в значение `true` позволяет агенту выполнять произвольный Python-код, сгенерированный LLM, во время выполнения на сервере

Генеративный ИИ. Этот код может выполнять любые операции, доступные Python, поэтому используйте эту опцию только с доверенными данными и в безопасной среде. Дополнительные сведения см. в описании уязвимости CVE-2024-21513. Этот компонент основан на **основном компоненте агента**.

Компонент создаёт CSV-агента из CSV-файла и языковой модели. Дополнительные сведения см. в документации LangChain CSV agent.

Параметры CSV-агента

Название	Тип	Описание
allow_dangerous_code	Boolean	Входной параметр. По умолчанию: false. Установите true, чтобы разрешить агенту выполнять произвольный Python-код. Дополнительные сведения см. в предупреждении о безопасности CSV-агента.
llm	LanguageModel	Входной параметр. Языковая модель для агента.
path	File	Входной параметр. Путь к CSV-файлу.
agent_type	String	Входной параметр. Тип создаваемого агента.
agent	AgentExecutor	Выходной параметр. Экземпляр CSV-агента.

Агент с инструментами OpenAI

Этот компонент основан на **основном компоненте агента**.

Компонент создаёт агента с инструментами OpenAI. Дополнительные сведения см. в документации LangChain OpenAI agent.

Параметры агента с инструментами OpenAI

Название	Тип	Описание
llm	LanguageModel	Входной параметр. Языковая модель для использования.
tools	List[Tool]	Входной параметр. Инструменты, к которым агент будет иметь доступ.
system_prompt	String	Входной параметр. Системный запрос, предоставляющий контекст агенту.
input_value	String	Входной параметр. Пользовательский ввод для агента.

Название	Тип	Описание
memory	Memory	Входной параметр. Память для сохранения контекста между вызовами агента.
max_iterations	Integer	Входной параметр. Максимальное количество итераций агента.
verbose	Boolean	Входной параметр. Выводить ли промежуточные шаги агента.
handle_parsing_errors	Boolean	Входной параметр. Обработать ли ошибки разбора в агенте.
agent	AgentExecutor	Выходной параметр. Экземпляр агента с инструментами OpenAI.
output	String	Выходной параметр. Результат выполнения агента по входным данным.

Агент OpenAPI

Этот компонент основан на **основном компоненте агента**.

Компонент создаёт агента для взаимодействия со службами OpenAPI. Дополнительные сведения см. в документации LangChain OpenAPI toolkit.

Параметры агента OpenAPI

Название	Тип	Описание
llm	LanguageModel	Входной параметр. Языковая модель для использования.
openapi_spec	String	Входной параметр. Спецификация OpenAPI для службы.
base_url	String	Входной параметр. Базовый URL для API.
headers	Dict	Входной параметр. Необязательные заголовки для запросов к API.
agent_executor_kwargs	Dict	Входной параметр. Необязательные параметры для исполнителя агента.
agent	AgentExecutor	Выходной параметр. Экземпляр агента OpenAPI.

Хаб промптов

Этот компонент получает промпты из LangChain Hub.

Как и в **шаблоне промпта**, к компоненту добавляются дополнительные поля для каждой переменной в промпте. Например, промпт по умолчанию efriis/my-first-prompt добавляет поля для profession и question.

Параметры хаба промптов

Название	Отображаемое название	Описание
langchain_api_key	API-ключ LangChain	Входной параметр. API-ключ LangChain для использования.
langchain_hub_prompt	Промпт из хаба LangChain	Входной параметр. Промпт из LangChain Hub для использования.
prompt	Сформированный промпт	Выходной параметр. Сформированное сообщение промпта, возвращаемое методом build_prompt.

SQL-агент

Этот компонент основан на **основном компоненте агента**.

Компонент создаёт агента для взаимодействия с базами данных SQL. Дополнительные сведения см. в документации LangChain SQL agent.

Параметры SQL-агента

Название	Тип	Описание
llm	LanguageModel	Входной параметр. Языковая модель для использования.
database	Database	Входной параметр. Соединение с базой данных SQL.
top_k	Integer	Входной параметр. Количество результатов, возвращаемых запросом SELECT.
use_tools	Boolean	Входной параметр. Использовать ли инструменты для выполнения запросов.
return_intermediate_step	Boolean	Входной параметр. Возвращать ли промежуточные шаги агента.
max_iterations	Integer	Входной параметр. Максимальное количество итераций агента.

Название	Тип	Описание
max_execution_time	Integer	Входной параметр. Максимальное время выполнения в секундах.
early_stopping_method	String	Входной параметр. Метод ранней остановки агента.
verbose	Boolean	Входной параметр. Выводить ли промежуточные мысли агента.
agent	AgentExecutor	Выходной параметр. Экземпляр SQL-агента.

База данных SQL

Компонент LangChain **База данных SQL** устанавливает соединение с базой данных SQL.

Этот компонент отличается от компонента **База данных SQL**, который выполняет SQL-запросы в базах данных, совместимых с SQLAlchemy.

Разделители текста

Пакет **LangChain** включает следующие компоненты разделителей текста:

- **Разделитель текстовых символов**
- **Рекурсивный разделитель текста по языкам**
- **Разделитель текста на естественном языке**
- **Рекурсивный разделитель текста по символам**
- **Семантический разделитель текста**

Агент вызова инструментов

Этот компонент основан на **основном компоненте агента**.

Компонент создаёт агента для структурированного вызова инструментов с различными языковыми моделями. Дополнительные сведения см. в документации LangChain по вызову инструментов.

Параметры агента вызова инструментов

Название	Тип	Описание
llm	LanguageModel	Входной параметр. Языковая модель для использования.
tools	List[Tool]	Входной параметр. Список инструментов, доступных агенту.
system_message	String	Входной параметр. Системное сообщение для агента.
return_intermediate_step	Boolean	Входной параметр. Возвращать ли промежуточные шаги агента.

Название	Тип	Описание
max_iterations	Integer	Входной параметр. Максимальное количество итераций агента.
max_execution_time	Integer	Входной параметр. Максимальное время выполнения в секундах.
early_stopping_method	String	Входной параметр. Метод ранней остановки агента.
verbose	Boolean	Входной параметр. Выводить ли промежуточные мысли агента.
agent	AgentExecutor	Выходной параметр. Экземпляр агента вызова инструментов.

XML-агент

Этот компонент основан на **основном компоненте агента**.

Компонент создаёт XML-агента с использованием LangChain. Агент использует XML-форматирование для инструкций инструментов LLM. Дополнительные сведения см. в документации LangChain XML agent.

Параметры XML-агента

Название	Тип	Описание
llm	LanguageModel	Входной параметр. Языковая модель для агента.
user_prompt	String	Входной параметр. Пользовательский шаблон промпта с инструкциями по XML-форматированию.
tools	List[Tool]	Входной параметр. Список инструментов, доступных агенту.
agent	AgentExecutor	Выходной параметр. Экземпляр XML-агента.

Другие компоненты LangChain

Пакет **LangChain** также включает следующие компоненты:

- Поддельные эмбединги
- Экстрактор HTML-ссылок
- Запускаемый исполнитель
- **Spider**: обход и парсинг веб-страниц

Устаревшие компоненты LangChain

Устаревшие компоненты больше не поддерживаются и могут быть удалены в будущих версиях. Вы можете продолжать использовать их в существующих потоках, однако

рекомендуется как можно скорее заменить их поддерживаемыми компонентами. Рекомендуемые замены указаны в баннере **Устаревшее** на компонентах в ваших потоках, а также в примечаниях к выпуску и документации Генеративный ИИ, когда это возможно.

Если вы не уверены, как заменить устаревший компонент, воспользуйтесь **поиском** по поставщику, сервису или названию компонента. Компонент мог быть объявлен устаревшим в пользу полностью нового компонента, похожего компонента или новой версии того же компонента в другой категории.

Если очевидной замены нет, рассмотрите возможность адаптации другого компонента под ваш случай использования. Например, многие **основные компоненты** предоставляют общую функциональность, поддерживающую несколько поставщиков и сценариев использования, — например, компонент **API-запрос**.

Если ни один из этих вариантов не подходит, вы можете использовать код устаревшего компонента для создания собственного пользовательского компонента или начать обсуждение по этому компоненту.

Чтобы не поощрять использование устаревших компонентов в новых потоках, они скрыты по умолчанию. В визуальном редакторе нажмите **Настройки компонентов**, чтобы переключить фильтр **Устаревшее**.

Следующие компоненты LangChain являются устаревшими:

- **Conversation Chain**
- **LLM Checker Chain**
- **LLM Math Chain**
- **Естественный язык в SQL**
- **Retrieval QA**
- **Поисковый модуль с самоформированием запроса**
- **JSON-агент**
- **Векторное хранилище Info/Agent**
- **VectorStoreRouterAgent**

Чтобы заменить эти компоненты, рассмотрите другие компоненты пакета **LangChain** или общие компоненты Генеративный ИИ, например компонент **Агент** или компонент **База данных SQL**.

26.37 LiteLLM

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Компонент пакета **LiteLLM** подключается к моделям через прокси LiteLLM, который перенаправляет запросы к нескольким поставщикам LLM. Использование прокси позволяет менять поставщиков моделей без изменения учётных данных в ваших потоках. Вы аутентифицируетесь в прокси с помощью единого ключа, а прокси использует собственные настроенные учётные данные для обращения к поставщикам. Виртуальные ключи создаются администратором прокси. Дополнительные сведения об управлении виртуальными ключами см. в разделе Virtual Keys документации LiteLLM.

Генерация текста через прокси LiteLLM

Компонент **LiteLLM Proxy** генерирует текст с использованием поставщика LLM.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда необходимо применять модель через прокси LiteLLM в качестве LLM для другого компонента, которым управляет LLM, — например, компонента **Агент** или **Умное преобразование**.

Дополнительные сведения см. в разделе Компоненты языковой модели.

Параметры прокси LiteLLM

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
api_base	String	Входной параметр. Базовый URL прокси LiteLLM. По умолчанию: http://localhost:4000/v1.
api_key	SecretString	Входной параметр. Виртуальный ключ для аутентификации в прокси LiteLLM.
model_name	String	Входной параметр. Название модели, например gpt-4o или claude-3-opus.
temperature	Float	Входной параметр. Управляет случайностью вывода. Более низкие значения дают более детерминированные результаты. Диапазон: [0.0, 2.0]. По умолчанию: 0.7.
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для снятия ограничения. Диапазон: [0, 128000]. Расширенный параметр.
timeout	Integer	Входной параметр. Тайм-аут запроса в секундах. По умолчанию: 60.
max_retries	Integer	Входной параметр. Максимальное количество повторных попыток при ошибке. По умолчанию: 2.

Название	Тип	Описание
stream	Boolean	Входной параметр. Включить потоковую передачу ответа.

26.38 LM Studio

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Компоненты в пакете **LM Studio** позволяют использовать модели из локального или удалённого экземпляра LM Studio. Для работы компонентов может потребоваться аутентификация с помощью API-ключа LM Studio. Дополнительные сведения о моделях LM Studio, подключениях и учётных данных см. в документации LM Studio.

Генерация текста LM Studio

Компонент **LM Studio** генерирует текст с использованием локальных языковых моделей LM Studio.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда необходимо применять модель LM Studio в качестве LLM для другого компонента, которым управляет LLM, — например, компонента **Агент** или **Умное преобразование**.

Дополнительные сведения см. в разделе Компоненты языковой модели.

Параметры генерации текста LM Studio

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
base_url	String	Входной параметр. URL-адрес сервера LM Studio. По умолчанию: http://localhost:1234.
api_key	SecretString	Входной параметр. API-ключ для аутентификации в LM Studio, если требуется.
max_tokens	Integer	Входной параметр. Максимальное количество токенов в ответе. По умолчанию: 512.
temperature	Float	Входной параметр. Управляет случайностью вывода. Диапазон: [0.0, 2.0]. По умолчанию: 0.7.
top_p	Float	Входной параметр. Управляет разнообразием вывода с помощью ядерной выборки (nucleus sampling).

Название	Тип	Описание
stop	List[String]	Диапазон: [0.0, 1.0]. По умолчанию: 1.0. Входной параметр. Список строк, при появлении которых генерация останавливается.
stream	Boolean	Входной параметр. Включить потоковую передачу ответа. По умолчанию: false.
presence_penalty	Float	Входной параметр. Штраф за повторение токенов, уже встречавшихся в тексте. Диапазон: [-2.0, 2.0]. По умолчанию: 0.0.
frequency_penalty	Float	Входной параметр. Штраф за часто встречающиеся токены. Диапазон: [-2.0, 2.0]. По умолчанию: 0.0.

Эмбединги LM Studio

Компонент **Эмбединги LM Studio** генерирует векторные представления (эмбединги) с использованием моделей LM Studio.

Дополнительные сведения об использовании компонентов моделей эмбедингов в потоках см. в разделе Компоненты моделей эмбедингов.

Параметры эмбедингов LM Studio

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
model	Модель	Входной параметр. Модель LM Studio для генерации эмбедингов.
base_url	Базовый URL LM Studio	Входной параметр. Базовый URL API LM Studio.
api_key	API-ключ LM Studio	Входной параметр. API-ключ для аутентификации в LM Studio.
temperature	Температура	Входной параметр. Параметр температуры для модели.

26.39 MariTalk

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **MariTalk**.

Дополнительные сведения см. в документации MariTalk.

Генерация текста MariTalk

Компонент **MariTalk** генерирует текст с использованием LLM MariTalk.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда необходимо применять модель MariTalk в качестве LLM для другого компонента, которым управляет LLM, — например, компонента **Агент** или **Умное преобразование**.

Дополнительные сведения см. в разделе Компоненты языковой модели.

Параметры генерации текста MariTalk

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для неограниченного вывода. По умолчанию: 512.
model_name	String	Входной параметр. Имя модели MariTalk. Варианты: sabia-2-small, sabia-2-medium. По умолчанию: sabia-2-small.
api_key	SecretString	Входной параметр. API-ключ MariTalk для аутентификации.
temperature	Float	Входной параметр. Управляет случайностью вывода. Диапазон: [0.0, 1.0]. По умолчанию: 0.5.
endpoint_url	String	Входной параметр. URL конечной точки API MariTalk. По умолчанию: https://api.maritalk.com.

26.40 Mem0

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Mem0**.

Память чата Mem0

Компонент **Память чата Mem0** извлекает и сохраняет сообщения чата, используя хранилище памяти Mem0.

Параметры памяти чата Mem0

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
mem0_config	Конфигурация Mem0	Входной параметр. Словарь конфигурации для инициализации экземпляра памяти Mem0.
ingest_message	Сообщение для приёма	Входной параметр. Содержимое сообщения, которое будет сохранено в памяти Mem0.
existing_memory	Существующий экземпляр памяти	Входной параметр. Необязательный существующий экземпляр памяти Mem0.
user_id	Идентификатор пользователя	Входной параметр. Идентификатор пользователя, связанного с сообщениями.
search_query	Поисковый запрос	Входной параметр. Текст для поиска связанных записей в памяти Mem0.
mem0_api_key	API-ключ Mem0	Входной параметр. API-ключ для платформы Mem0. Оставьте пустым для использования локальной версии.
metadata	Метаданные	Входной параметр. Дополнительные метаданные, связанные с сохраняемым сообщением.
openai_api_key	API-ключ OpenAI	Входной параметр. API-ключ для OpenAI. Требуется при использовании эмбеддингов OpenAI без предоставленной конфигурации.

Выходы памяти чата Mem0

Компонент **Память чата Mem0** может возвращать либо **Память Mem0** (Memory), либо **Результаты поиска** (Data). Выберите нужный тип рядом с портом выхода компонента.

Используйте **Память чата Mem0** для операций хранения и извлечения памяти совместно с компонентом **История сообщений**.

Используйте выход **Результаты поиска**, чтобы получить конкретные записи памяти по поисковому запросу.

26.41 Milvus

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Milvus**.

Векторное хранилище Milvus

Компонент **Milvus** читает и записывает данные в векторные хранилища Milvus, используя экземпляр векторного хранилища Milvus.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ построен на основе LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента экземпляр обычно создаётся под именем `vector_store`, однако некоторые компоненты используют другое имя — как правило, название поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика, эти параметры могут использовать значения по умолчанию или позволять переопределение через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Дополнительные сведения о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в вашем потоке в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба формата, выберите нужный рядом с портом выхода компонента векторного хранилища в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры векторного хранилища Milvus

Параметры компонента векторного хранилища позволяют подробнее узнать о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после задания других параметров или выбора определённых значений. Условные параметры могут отсутствовать на панели **Управление** до тех пор, пока не установлены необходимые зависимости.

Допустимые значения и описание функциональности см. в документации Milvus или в коде компонента.

Название	Тип	Описание
collection_name	String	Входной параметр. Название коллекции Milvus.
collection_description	String	Входной параметр. Описание коллекции Milvus.
uri	String	Входной параметр. URI подключения к Milvus.
password	SecretString	Входной параметр. Пароль для подключения к Milvus.
username	SecretString	Входной параметр. Имя пользователя для подключения к Milvus.
batch_size	Integer	Входной параметр. Количество записей, обрабатываемых за один пакет.
search_query	String	Входной параметр. Запрос для поиска по сходству.
ingest_data	Данные	Входной параметр. Данные для загрузки в векторное хранилище.
embedding	Эмбединги	Входной параметр. Функция эмбединга для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.
search_type	String	Входной параметр. Тип выполняемого поиска.
search_score_threshold	Float	Входной параметр. Минимальный порог оценки сходства для результатов поиска.
search_filter	Dict	Входной параметр. Фильтры метаданных для поискового запроса.
setup_mode	String	Входной параметр. Режим конфигурации при настройке векторного хранилища.
vector_dimensions	Integer	Входной параметр. Размерность векторов.

Название	Тип	Описание
pre_delete_collection	Boolean	Входной параметр. Удалять ли коллекцию перед созданием новой.

26.42 MistralAI

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **MistralAI**.

Дополнительные сведения о функциях и возможностях MistralAI, используемых этими компонентами, см. в документации MistralAI.

Генерация текста MistralAI

Компонент **MistralAI** генерирует текст с использованием LLM MistralAI.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда необходимо применять модель MistralAI в качестве LLM для другого компонента, которым управляет LLM, — например, компонента **Агент** или **Умное преобразование**.

Дополнительные сведения см. в разделе Компоненты языковой модели.

Параметры генерации текста MistralAI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для неограниченного вывода.
model_name	String	Входной параметр. Имя модели Mistral AI. Доступные варианты: open-mixtral-8x7b, open-mixtral-8x22b, mistral-small-latest, mistral-medium-latest, mistral-large-latest, codestral-latest. По умолчанию: codestral-latest.
mistral_api_base	String	Входной параметр. Базовый URL API Mistral. По умолчанию: https://api.mistral.ai/v1 .
api_key	SecretString	Входной параметр. API-ключ Mistral для аутентификации.

Название	Тип	Описание
temperature	Float	Входной параметр. Управляет случайностью вывода. По умолчанию: 0.5.
max_retries	Integer	Входной параметр. Максимальное количество повторных попыток при неудачных запросах к API. По умолчанию: 5.
timeout	Integer	Входной параметр. Тайм-аут запросов к API в секундах. По умолчанию: 60.
max_concurrent_requests	Integer	Входной параметр. Максимальное количество одновременных запросов к API. По умолчанию: 3.
top_p	Float	Входной параметр. Параметр ядерной выборки (nucleus sampling). По умолчанию: 1.
random_seed	Integer	Входной параметр. Начальное значение генератора случайных чисел. По умолчанию: 1.
safe_mode	Boolean	Входной параметр. Включает безопасный режим генерации контента.

Эмбединги MistralAI

Компонент **Эмбединги MistralAI** генерирует векторные представления (эмбединги) с использованием моделей MistralAI.

Дополнительные сведения об использовании компонентов моделей эмбедингов в потоках см. в разделе Компоненты моделей эмбедингов.

Параметры эмбедингов MistralAI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
model	String	Входной параметр. Модель MistralAI для использования. По умолчанию: mistral-embed.
mistral_api_key	SecretString	Входной параметр. API-ключ для аутентификации в MistralAI.

Название	Тип	Описание
max_concurrent_requests	Integer	Входной параметр. Максимальное количество одновременных запросов к API. По умолчанию: 64.
max_retries	Integer	Входной параметр. Максимальное количество повторных попыток при неудачных запросах. По умолчанию: 5.
timeout	Integer	Входной параметр. Тайм-аут запроса в секундах. По умолчанию: 120.
endpoint	String	Входной параметр. Пользовательский URL конечной точки API. По умолчанию: https://api.mistral.ai/v1/ .

26.43 MongoDB

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **MongoDB**.

MongoDB Atlas

Компонент **MongoDB Atlas** читает и записывает данные в векторные хранилища MongoDB Atlas с использованием экземпляра `MongoDBAtlasVectorSearch`.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента экземпляр обычно создаётся под именем `vector_store`, однако некоторые компоненты используют другое имя — как правило, название поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика, эти параметры могут использовать значения по умолчанию или позволять переопределение через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Для получения сведений о конкретных параметрах см. справочник API LangChain и документацию поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в вашем потоке в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба формата, выберите нужный рядом с портом выхода компонента векторного хранилища в визуальном редакторе.

Параметры MongoDB Atlas

Параметры компонента векторного хранилища позволяют подробнее узнать о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и доступны только после задания других параметров или выбора определённых значений. Условные параметры могут отсутствовать на панели **Управление** до тех пор, пока не установлены необходимые зависимости.

Допустимые значения и описание функциональности см. в документации MongoDB Atlas или в коде компонента.

Название	Тип	Описание
mongodb_atlas_cluster_uri	SecretString	Входной параметр. URI подключения к кластеру MongoDB Atlas. Обязательный.
enable_mtls	Boolean	Входной параметр. Включить взаимную аутентификацию TLS (mTLS). По умолчанию: false.
mongodb_atlas_client_certificate	SecretString	Входной параметр. Клиентский сертификат, объединённый с закрытым ключом для аутентификации mTLS. Обязательный, если mTLS включён.
db_name	String	Входной параметр. Имя базы данных. Обязательный.
collection_name	String	Входной параметр. Имя коллекции. Обязательный.
index_name	String	Входной параметр. Имя индекса Atlas Search — должен быть индексом векторного поиска. Обязательный.
insert_mode	String	Входной параметр. Способ вставки новых документов в коллекцию. Варианты: append или overwrite. По умолчанию: append.

Название embedding	Тип Эмбединги	Описание Входной параметр. Модель векторного представления для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске по сходству. По умолчанию: 4.
index_field	String	Входной параметр. Поле для индексации. По умолчанию: embedding.
filter_field	String	Входной параметр. Поле для фильтрации индекса.
number_dimensions	Integer	Входной параметр. Размерность вектора эмбединга. По умолчанию: 1536.
similarity	String	Входной параметр. Метод измерения сходства между векторами. Варианты: cosine, euclidean или dotProduct. По умолчанию: cosine.
quantization	String	Входной параметр. Квантование снижает потребление памяти, преобразуя 32-битные числа с плавающей точкой в меньший тип данных. Варианты: scalar или binary.

26.44 Notion

Чтобы использовать компоненты Notion в модуле «Генеративный ИИ», сначала необходимо создать интеграцию Notion и настроить её с необходимыми возможностями. В этом руководстве показан пошаговый процесс настройки интеграции Notion и предоставления ей доступа к базам данных Notion, а также приведены два примера потоков с компонентами Notion.

Настройка приложения Notion

Для создания приложения Notion вам потребуются:

- Учётная запись Notion с доступом к рабочему пространству, где вы хотите использовать интеграцию.

- Права администратора в рабочем пространстве Notion для создания и управления интеграциями.

Создание интеграции Notion

1. Перейдите на страницу интеграций Notion.
2. Нажмите **Новая интеграция**.
3. Введите имя интеграции и выберите рабочее пространство, где вы хотите её использовать.
4. Опционально: включите необходимые возможности интеграции в соответствии с вашими требованиями.
5. Нажмите **Подтвердить**, чтобы создать интеграцию.

Настройка возможностей интеграции

После создания интеграции настройте её возможности, чтобы определить, какие действия она может выполнять и к каким данным получать доступ.

1. На странице настроек интеграции перейдите на вкладку **Возможности**.
2. Включите необходимые возможности. Например:
 - Если интеграции нужно читать данные из Notion, включите **Чтение содержимого**.
 - Если интеграции нужно создавать контент в Notion, включите **Вставка содержимого**.
 - Если интеграции нужно обновлять существующий контент в Notion, включите **Обновление содержимого**.
3. Настройте уровень доступа к пользовательской информации в соответствии с требованиями вашей интеграции.
4. Сохраните изменения.

Получение токена интеграции

Токен интеграции — конфиденциальная информация. Храните его в безопасности и никогда не публикуйте. Сохраняйте токен в надёжном месте: в конфигурации Генеративный ИИ или в переменных окружения. Для аутентификации интеграции с Notion необходимо получить токен интеграции.

1. На странице настроек интеграции перейдите на вкладку **Секреты**.
2. Скопируйте значение **Токен внутренней интеграции**. Этот токен используется для аутентификации вашей интеграции с Notion.

Предоставление интеграции доступа к базам данных Notion

Чтобы интеграция могла взаимодействовать с базами данных Notion, необходимо предоставить ей доступ к конкретным базам данных:

1. Откройте базу данных Notion, к которой вы хотите предоставить доступ.
2. Нажмите **Поделиться**.
3. В разделе **Пригласить** выберите вашу интеграцию из списка.
4. Нажмите **Пригласить**, чтобы предоставить интеграции доступ к базе данных.

Если ваша база данных содержит ссылки на другие базы данных, необходимо также предоставить интеграции доступ к ним.

Повторите этот шаг для каждой базы данных, к которой должна иметь доступ ваша интеграция.

Компоненты Notion в модуле «Генеративный ИИ»

- **Добавить контент на страницу:** преобразует текст в формате Markdown в блоки Notion и добавляет их в указанную страницу.
- **Создать страницу:** создаёт новую страницу в указанной базе данных Notion с заданными свойствами.
- **Список свойств базы данных:** получает свойства указанной базы данных Notion.
- **Список страниц:** выполняет запрос к базе данных Notion с параметрами фильтрации и сортировки.
- **Список пользователей:** получает список пользователей из рабочего пространства Notion.
- **Просмотр содержимого страницы:** получает содержимое страницы Notion в виде обычного текста.
- **Поиск:** ищет все страницы и базы данных, к которым у интеграции есть доступ. Можно фильтровать результаты по страницам или базам данных и задавать направление сортировки.
- **Обновить свойство страницы:** обновляет свойства существующей страницы Notion.

Каждый из этих компонентов может возвращать типы данных Data и Tool (подробнее о типах данных).

Агент заметок встреч Notion

Агент заметок встреч Notion — это AI-инструмент, который автоматически обрабатывает стенограммы встреч и обновляет рабочее пространство Notion. Он определяет задачи, действия и ключевые моменты встреч, а затем создаёт новые задачи или обновляет существующие в Notion без ручного ввода.

Для создания этого потока вам потребуются:

- Приложение Notion
- Ключ API Notion
- Ключ API OpenAI
- Рекомендуется: скачайте JSON-файл потока Conversational Notion Agent, а затем импортируйте поток в модуле «Генеративный ИИ».

Обращайтесь со всеми ключами API и другими учётными данными как с конфиденциальной информацией. Используйте безопасные хранилища и не публикуйте их.

Компоненты потока

- **Стенограмма встречи** (текстовый ввод): позволяет пользователю ввести стенограмму встречи непосредственно в поток.
- **Список пользователей** (компонент Notion): получает список пользователей из рабочего пространства Notion. Требуется ключ API Notion.
- **Список баз данных** (компонент Notion): ищет и возвращает список всех баз данных в рабочем пространстве Notion. Требуется:
 - Ключ API Notion.
 - Запрос (необязательно).
 - Тип фильтра (по умолчанию: база данных).
 - Направление сортировки.
- **Шаблон промпта:** создаёт динамический шаблон запроса, используя следующие входные данные:
 - Стенограмма встречи.
 - Список пользователей.

- Список баз данных.
- Текущая дата.
- **Сводка встречи** (агент с вызовом инструментов): анализирует стенограмму и определяет задачи и действия. Требуется:
 - Системный промпт (из компонента **Шаблон промпта**).
 - Языковую модель (OpenAI).
 - Инструменты:
 - **Поиск Notion**: ищет страницы и базы данных.
 - **Список свойств базы данных**: получает свойства указанной базы данных. Требуется идентификатор базы данных и ключ API Notion.
 - **Создать страницу**: создаёт новую страницу в базе данных Notion. Требуется идентификатор базы данных, свойства (JSON) и ключ API Notion.
 - **Обновить свойство страницы**: обновляет свойства существующей страницы Notion. Требуется идентификатор страницы, ключ API Notion и обновляемые свойства.
 - **Добавить контент на страницу**: преобразует текст Markdown в блоки Notion и добавляет их на указанную страницу. Требуется идентификатор страницы/блока, ключ API Notion и текст Markdown.
- **Агент Notion** (агент с вызовом инструментов): выполняет действия в Notion на основе сводки встречи. Требуется:
 - Системный промпт (из второго компонента **Шаблон промпта**).
 - Языковую модель (OpenAI).
 - Инструменты:
 - **Список свойств базы данных**: требует идентификатор базы данных и ключ API Notion.
 - **Создать страницу**: требует идентификатор базы данных, свойства (JSON) и ключ API Notion.
 - **Обновить свойство страницы**: требует идентификатор страницы, ключ API Notion и обновляемые свойства.
 - **Добавить контент на страницу**: требует идентификатор страницы/блока, ключ API Notion и текст Markdown.
- **Вывод чата**: отображает итоговый результат агента Notion в Playground.

Процесс потока

1. Пользователь вводит стенограмму встречи.
2. Поток получает список пользователей и баз данных Notion.
3. Формируется запрос на основе стенограммы, списка пользователей, списка баз данных и текущей даты.
4. Сводка встречи анализирует стенограмму и определяет задачи и действия.
5. Агент Notion использует сводку встречи для:
 - создания новых страниц для новых задач;
 - обновления существующих страниц для текущих задач;
 - добавления контента на страницы с заметками встречи.
6. Вывод чата отображает сводку действий, выполненных в Notion.

Запуск потока заметок встречи

1. Создайте поток вручную или импортируйте готовый JSON-файл:
 - Рекомендуется: скачайте JSON-файл потока Meeting Notes Agent, а затем импортируйте поток в модуле «Генеративный ИИ».
 - Либо создайте пустой поток, добавьте описанные выше компоненты и соедините их, как показано на диаграмме потока.
2. Введите ключи API Notion и OpenAI в соответствующих компонентах.
3. Вставьте стенограмму встречи в компонент **Стенограмма встречи**.
Для оптимальных результатов используйте подробные стенограммы: качество вывода зависит от полноты входных данных.
4. Запустите поток, нажав **Run component** в компоненте **Вывод чата**, или откройте **Playground**.
5. Просмотрите результат — сводку действий, выполненных в вашем рабочем пространстве Notion.
6. Опционально: настройте поток под потребности вашей команды:
 - Измените системный промпт, чтобы скорректировать поведение или контекст агента.
 - Добавьте или удалите инструменты Notion по мере необходимости.
 - Измените параметры модели OpenAI (например, temperature), чтобы настроить стиль ответов.

Разговорный агент Notion

Разговорный агент Notion — это AI-ассистент, который взаимодействует с рабочим пространством Notion через диалог на естественном языке. Поток выполняет задачи Notion — создание страниц, поиск информации и управление контентом — через чат-интерфейс.

Для создания этого потока вам потребуются:

- Приложение Notion
- Учётная запись Notion и ключ API
- Ключ API OpenAI
- Рекомендуется: скачайте JSON-файл потока Conversational Notion Agent, а затем импортируйте поток в модуле «Генеративный ИИ».

Компоненты потока

- **Ввод в чат:** принимает запросы и команды пользователя.
- **Вывод чата:** отображает ответы агента.
- **Языковая модель:** обрабатывает ввод пользователя и генерирует ответы с помощью модели OpenAI.
- **Агент с вызовом инструментов:** координирует использование инструментов Notion на основе запроса пользователя.
- **Инструменты:** объединяет несколько специализированных инструментов Notion в один набор.
- **Инструменты Notion:** компоненты для взаимодействия с Notion, включая:
 - Список пользователей
 - Список свойств базы данных
 - Список страниц
 - Просмотр содержимого страницы
 - Создать страницу
 - Обновить свойство страницы

- Добавить контент на страницу
- Поиск
- **История сообщений:** хранит историю разговора.
- **Шаблон промпта:** предоставляет системные инструкции и контекст для агента.
- **Текущая дата:** предоставляет текущие дату и время для контекста.

Запуск разговорного агента Notion

1. Откройте Генеративный ИИ и создайте новый поток.
2. Добавьте перечисленные выше компоненты на холст потока, либо скачайте JSON-файл потока Conversational Notion Agent и импортируйте его в модуле «Генеративный ИИ».
3. Подключите компоненты, как показано на диаграмме потока.
4. Введите ключи API Notion и OpenAI в соответствующих компонентах.
5. Чтобы запустить поток, нажмите кнопку воспроизведения в модуле «Генеративный ИИ».
6. В чат-интерфейсе введите запрос, связанный с Notion. Например:
 - «Создать новую страницу в моей базе данных»
 - «Найти страницы об управлении проектами»
 - «Показать всех пользователей в моём рабочем пространстве Notion»
7. Агент обработает запрос, применит нужные инструменты Notion и предоставит ответ.
8. Продолжайте разговор, задавая уточняющие вопросы или давая новые команды.
9. Опционально: настройте поток под потребности вашей команды:
 - Скорректируйте системный промпт, чтобы изменить поведение или контекст агента.
 - Добавьте или удалите инструменты Notion по мере необходимости.
 - Измените параметры модели OpenAI (например, temperature), чтобы настроить стиль ответов.

Устранение неполадок

Если вы столкнулись с проблемами:

1. Убедитесь, что все ключи API правильно настроены и имеют необходимые разрешения.
2. Проверьте, что интеграция Notion имеет доступ к нужным страницам и базам данных.
3. Убедитесь, что все компоненты правильно подключены в потоке.
4. Просмотрите журналы Генеративный ИИ на наличие сообщений об ошибках.

Для более продвинутого использования и вариантов интеграции см. документацию Notion API и документацию Генеративный ИИ.

26.45 Novita

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Novita**.

Novita AI

Этот компонент генерирует текст с использованием языковых моделей Novita.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда хотите применить модель Novita в качестве LLM для другого компонента, управляемого LLM, например **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. компоненты языковых моделей.

Параметры Novita AI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Изменить все параметры компонента можно через панель инспекции компонента, которая появляется при его выборе.

Название	Тип	Описание
api_key	SecretString	Входной параметр. Ваш ключ API Novita AI.
model	String	Входной параметр. Идентификатор используемой модели Novita AI.
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Значение 0 означает отсутствие ограничений.
temperature	Float	Входной параметр. Управляет случайностью вывода. Диапазон: [0.0, 1.0]. По умолчанию: 0.7.
top_p	Float	Входной параметр. Управляет ядерной выборкой (nucleus sampling). Диапазон: [0.0, 1.0]. По умолчанию: 1.0.
frequency_penalty	Float	Входной параметр. Штраф за частоту повторения токенов. Диапазон: [0.0, 2.0]. По умолчанию: 0.0.
presence_penalty	Float	Входной параметр. Штраф за наличие уже встречавшихся токенов. Диапазон: [0.0, 2.0]. По умолчанию: 0.0.

26.46 NVIDIA

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

На этой странице описаны компоненты, доступные в пакете **NVIDIA**.

Генерация текста NVIDIA

Этот компонент генерирует текст с использованием языковых моделей NVIDIA. Для получения дополнительной информации о языковых моделях NVIDIA см. документацию NVIDIA AI.

Параметры генерации текста NVIDIA

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Значение 0 означает отсутствие ограничений.
model_name	String	Входной параметр. Имя используемой модели NVIDIA. По умолчанию: mistralai/mixtral-8x7b-instruct-v0.1.
base_url	String	Входной параметр. Базовый URL API NVIDIA. По умолчанию: https://integrate.api.nvidia.com/v1 .
nvidia_api_key	SecretString	Входной параметр. Ключ API NVIDIA для аутентификации.
temperature	Float	Входной параметр. Управляет случайностью вывода. По умолчанию: 0.1.
seed	Integer	Входной параметр. Определяет начальное состояние генератора для воспроизводимости результатов. По умолчанию: 1.
model	LanguageModel	Выходной параметр. Экземпляр ChatNVIDIA, настроенный с указанными параметрами.

NVIDIA NIM на WSL2

NVIDIA NIM (NVIDIA Inference Microservices) предоставляет контейнеры для самостоятельного размещения микросервисов инференса с ускорением GPU.

Вы можете использовать компонент **NVIDIA** для подключения Генеративный ИИ к NVIDIA NIM на системе RTX Windows с установленным Windows Subsystem for Linux 2 (WSL2). В следующем примере компонент **NVIDIA** в модуле «Генеративный ИИ» подключается к развёрнутому mistral-nemo-12b-instruct NIM на системе **RTX Windows** с **WSL2**.

1. Подготовьте вашу систему:
 - Установите NVIDIA NIM для WSL2.

- Разверните контейнер NIM согласно инструкциям для выбранной модели. Требования различаются в зависимости от модели. Например, для развёртывания `mistral-nemo-12b-instruct` NIM следуйте инструкциям для **Windows on RTX AI PCs (Beta)** в обзоре развёртывания модели.
 - Windows 11 сборки 23H2 или новее.
 - Не менее 12 ГБ ОЗУ.
2. Создайте поток на основе шаблона **Основные промпты**.
 3. Замените компонент **OpenAI** на компонент **NVIDIA**.
 4. В поле **Базовый URL** компонента **NVIDIA** укажите URL, по которому доступен ваш NIM. Если вы следовали инструкциям по развёртыванию модели, значение будет `http://localhost:8000/v1`.
 5. В поле **Ключ API NVIDIA** укажите ваш ключ API NVIDIA.
 6. Выберите модель в поле **Название модели**.
 7. Откройте **Playground** и начните диалог с вашей моделью NIM.

Эмбединги NVIDIA

Компонент **Эмбединги NVIDIA** генерирует эмбединги с использованием моделей NVIDIA.

Для получения дополнительной информации об использовании компонентов моделей эмбедингов в потоках см. компоненты моделей эмбедингов.

Параметры эмбедингов NVIDIA

Название	Тип	Описание
<code>model</code>	String	Входной параметр. Модель NVIDIA для генерации эмбедингов, например <code>nvidia/nv-embed-v1</code> .
<code>base_url</code>	String	Входной параметр. Базовый URL API NVIDIA. По умолчанию: <code>https://integrate.api.nvidia.com/v1</code> .
<code>nvidia_api_key</code>	SecretString	Входной параметр. Ключ API для аутентификации в сервисе NVIDIA.
<code>temperature</code>	Float	Входной параметр. Температура модели при генерации эмбедингов. По умолчанию: 0.1.
<code>embeddings</code>	Эмбединги	Выходной параметр. Экземпляр <code>NVIDIAEmbeddings</code> для генерации эмбедингов.

Учитывайте ограничение размера фрагментов для вашей модели эмбедингов: ошибки токенизации могут возникнуть, если фрагменты текста слишком велики. Для получения дополнительной информации см. ошибки токенизации из-за размера фрагмента.

Ранжирование NVIDIA

Этот компонент выполняет поиск и переранжирование документов с использованием API NVIDIA.

Извлечение данных NVIDIA

Компонент **Извлечение данных NVIDIA** интегрируется с микросервисом NVIDIA nv-ingest для загрузки, обработки и извлечения данных из файлов.

Сервис nv-ingest поддерживает несколько методов извлечения для файлов PDF, DOCX и PPTX, а также включает пред- и постобработку: разбиение на фрагменты и генерацию эмбедингов. Режим высокого разрешения использует метод nemoretriever-parse для более качественного извлечения из отсканированных PDF-документов. Эта функция доступна только для PDF-файлов.

Компонент **Извлечение данных NVIDIA** использует клиент NVIDIA Ingestor, загружает файлы через конечную точку NVIDIA Ingest и возвращает обработанное содержимое в виде списка объектов Data. Ingestor принимает дополнительные параметры конфигурации для извлечения данных из других текстовых форматов. Подробнее см. в разделе Параметры.

Компонент **Извлечение данных NVIDIA** также известен как NV-Ingest и NeMo Retriever Extraction.

Использование компонента в потоке

Компонент **Извлечение данных NVIDIA** принимает Message и возвращает Data. Компонент вызывает конечную точку микросервиса NVIDIA Ingest для загрузки локального файла и извлечения текста.

Чтобы использовать компонент **Извлечение данных NVIDIA** в потоке, выполните следующие шаги:

1. Подготовьте вашу систему:

- Настройте конечную точку NVIDIA Ingest. Подробнее см. в быстром старте NVIDIA Ingest.
- Установите дополнительные зависимости в вашей среде Генеративный ИИ. Чтобы установить их в виртуальном окружении, выполните одну из следующих команд.

- Если вы клонировали репозиторий Генеративный ИИ и установили его из исходного кода:

```
source /bin/activate
uv sync --extra nv-ingest
uv run astra run
```

- Если вы установили Генеративный ИИ из Python Package Index:

```
source /bin/activate
uv pip install --prerelease=allow 'astra[nv-ingest]'
uv run astra run
```

2. Добавьте компонент **Извлечение данных NVIDIA** в ваш поток.
3. В поле **Базовый URL** введите URL конечной точки NVIDIA Ingest. Вы также можете сохранить URL как глобальную переменную для повторного использования в нескольких компонентах и потоках.
4. Нажмите **Выберите файлы**, чтобы выбрать файл для загрузки.
5. Выберите тип содержимого для извлечения: текст, диаграммы, таблицы, изображения или инфографику.
6. Опционально: для PDF-файлов включите **Режим высокого разрешения** для более качественного извлечения из отсканированных документов.

7. Выберите, нужно ли разбивать текст на фрагменты.
Некоторые параметры скрыты по умолчанию в визуальном редакторе. Изменить все параметры компонента можно через панель инспекции компонента, которая появляется при его выборе.
8. Нажмите **Запустить компонент**, чтобы загрузить файл. Затем нажмите **Журналы** или **Проверка вывода**, чтобы убедиться в успешной загрузке.
9. Чтобы сохранить обработанные данные в векторной базе данных, добавьте компонент векторного хранилища и подключите выход Data компонента **Извлечение данных NVIDIA** к его входу.
После запуска потока обработанные данные сохранятся в векторной базе данных, и вы сможете выполнять к ней запросы.

Параметры

Компонент **Извлечение данных NVIDIA** имеет следующие параметры.

Для получения дополнительной информации см. документацию NV-Ingest.

Входные параметры

Название	Отображаемое название	Описание
base_url	URL приёма данных NVIDIA	URL API приёма данных NVIDIA.
path	Путь	Путь к файлу для обработки.
extract_text	Извлечь текст	Извлекать текст из документов. По умолчанию: true.
extract_charts	Извлечь диаграммы	Извлекать текст из диаграмм. По умолчанию: false.
extract_tables	Извлечь таблицы	Извлекать текст из таблиц. По умолчанию: true.
extract_images	Извлечь изображения	Извлекать изображения из документа. По умолчанию: true.
extract_infographics	Извлечь инфографику	Извлекать инфографику из документа. По умолчанию: false.
text_depth	Глубина текста	Уровень извлечения текста. Варианты: document, page, block, line, span. По умолчанию: page.
split_text	Разбить текст	Разбивать текст на фрагменты. По умолчанию: true.
chunk_size	Размер фрагмента	Количество токенов в фрагменте. По умолчанию: 500. Убедитесь, что размер фрагмента совместим с

Название	Отображаемое название	Описание
		вашей моделью эмбеддингов. Подробнее см. ошибки токенизации из-за размера фрагмента.
chunk_overlap	Перекрытие фрагментов	Количество токенов, перекрывающих предыдущий фрагмент. По умолчанию: 150.
filter_images	Фильтровать изображения	Фильтровать изображения (критерии фильтрации задаются в расширенных параметрах). По умолчанию: false.
min_image_size	Минимальный размер изображения	Минимальная ширина и высота изображения в пикселях. По умолчанию: 128.
min_aspect_ratio	Минимальное соотношение сторон	Минимально допустимое соотношение сторон (ширина / высота). По умолчанию: 0.2.
max_aspect_ratio	Максимальное соотношение сторон	Максимально допустимое соотношение сторон (ширина / высота). По умолчанию: 5.0.
dedup_images	Дедупликация изображений	Удалять дублирующиеся изображения. По умолчанию: true.
caption_images	Подписи к изображениям	Генерировать подписи к изображениям с помощью модели NVIDIA. По умолчанию: true.
high_resolution	Высокое разрешение (только PDF)	Обрабатывать PDF в режиме высокого разрешения для более качественного извлечения из отсканированных документов. По умолчанию: false.

Выходные данные

Компонент **Извлечение данных NVIDIA** возвращает список объектов Data, каждый из которых содержит:

- text: извлечённое содержимое.

- Для текстовых документов: извлечённый текст.
- Для таблиц и диаграмм: содержимое таблицы или диаграммы.
- Для изображений: подпись к изображению.
- Для инфографики: содержимое инфографики.
- `file_path`: имя и путь исходного файла.
- `document_type`: тип документа — `text`, `structured` или `image`.
- `description`: дополнительное описание содержимого.

Структура вывода зависит от `document_type`:

- Документы с `document_type`: `"text"` содержат:
 - Необработанный текст, извлечённый из документов, например абзацы из PDF или DOCX.
 - Содержимое хранится в поле `text`.
 - Данные извлекаются с помощью параметра `extract_text`.
- Документы с `document_type`: `"structured"` содержат:
 - Текст из таблиц, диаграмм и инфографики, обработанный с сохранением структурной информации.
 - Данные извлекаются с помощью параметров `extract_tables`, `extract_charts` и `extract_infographics`.
 - Содержимое хранится в поле `text` после обработки из метаданных `table_content`.
- Документы с `document_type`: `"image"` содержат:
 - Содержимое изображений, извлечённых из документов.
 - Подпись хранится в поле `text`, если включён параметр `caption_images`.
 - Данные извлекаются с помощью параметра `extract_images`.

Системный ассистент NVIDIA

Компонент **Системный ассистент NVIDIA** интегрирует потоки с NVIDIA G-Assist, позволяя взаимодействовать с драйверами GPU NVIDIA с помощью запросов на естественном языке. Например, спросите G-Assist: "Какова текущая температура моей видеокарты?" или "Сколько видеопамати сейчас доступно?" — чтобы получить информацию, а затем попросите изменить настройки GPU. Для получения дополнительной информации см. репозиторий NVIDIA G-Assist.

1. Подготовьте вашу систему:
 - Компонент **Системный ассистент NVIDIA** требует GPU NVIDIA под управлением Windows.
 - Он использует пакет `gassist.rise`, который устанавливается во всех версиях Генеративный ИИ, включающих этот компонент.
2. Создайте поток с компонентами **Ввод в чат**, **Системный ассистент NVIDIA** и **Вывод чата**.
 Это упрощённый пример, использующий только три компонента. В зависимости от вашего сценария поток может включать дополнительные компоненты или другие источники ввода и вывода.
3. Подключите компонент **Ввод в чат** к входу **Запрос** компонента **Системный ассистент NVIDIA**.

Параметр **Запрос** принимает запрос на естественном языке, который обрабатывается ассистентом NVIDIA G-Assist. В этом примере запрос поступает из компонента ввода

чата. Вы также можете ввести запрос напрямую в поле **Запрос** или подключить другой входной компонент.

4. Подключите выход компонента **Системный ассистент NVIDIA** к компоненту **Вывод чата**.
5. Чтобы протестировать поток, откройте **Playground** и задайте вопрос о вашем GPU, например: "What is my current GPU temperature?".
NVIDIA G-Assist запросит информацию о GPU и выведет ответ в **Playground**.
Выход компонента — объект Message с ответом NVIDIA G-Assist. Текстовый результат доступен в поле text объекта Message.

26.47 Ollama

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Ollama**.

Для получения дополнительной информации о функциях и возможностях Ollama, используемых этими компонентами, см. документацию Ollama.

Генерация текста Ollama

Этот компонент генерирует текст с использованием языковых моделей Ollama.

Чтобы использовать компонент **Ollama** в потоке, подключите Генеративный ИИ к локально запущенному серверу Ollama и выберите модель:

1. Добавьте компонент **Ollama** в ваш поток.
2. В поле **Базовый URL** введите адрес вашего локально запущенного сервера Ollama.
Это значение задаётся переменной окружения OLLAMA_HOST в Ollama. Базовый URL по умолчанию — http://127.0.0.1:11434.
3. После установления соединения выберите модель в поле **Название модели**, например llama3.2:latest.
Чтобы обновить список моделей сервера, нажмите **Обновить**.
4. Опционально: чтобы настроить дополнительные параметры, такие как температура или максимальное количество токенов, нажмите на компонент и откройте панель инспекции компонента.
5. Подключите компонент **Ollama** к другим компонентам потока в зависимости от того, как вы хотите использовать модель.

Компоненты языковых моделей могут возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel). Используйте выход **Языковая модель**, когда хотите применить модель Ollama в качестве LLM для другого компонента, управляемого LLM, например **Агент** или **Умное преобразование**. Для получения дополнительной информации см. компоненты языковых моделей.

В следующем примере поток использует выход LanguageModel, чтобы применить модель Ollama в качестве LLM для компонента **Агент**.

Эмбединги Ollama

Компонент **Эмбединги Ollama** генерирует эмбединги с использованием моделей эмбедингов Ollama.

Чтобы использовать этот компонент в потоке, подключите Генеративный ИИ к локально запущенному серверу Ollama и выберите модель эмбедингов:

1. Добавьте компонент **Эмбединги Ollama** в ваш поток.

2. В поле **Базовый URL Ollama** введите адрес вашего локально запущенного сервера Ollama.
Это значение задаётся переменной окружения OLLAMA_HOST в Ollama. Базовый URL по умолчанию — `http://127.0.0.1:11434`.
3. После установления соединения выберите модель в поле **Модель Ollama**, например `all-minilm:latest`.
Чтобы обновить список моделей сервера, нажмите **Обновить**.
4. Опционально: чтобы настроить дополнительные параметры, нажмите на компонент и откройте панель инспекции компонента. Доступные параметры зависят от выбранной модели.
5. Подключите компонент **Эмбединги Ollama** к другим компонентам потока. Для получения дополнительной информации об использовании компонентов моделей эмбедингов в потоках см. компоненты моделей эмбедингов.
В этом примере компонент **Эмбединги Ollama** генерирует эмбединги для текстовых фрагментов, извлечённых из PDF-файла, а затем сохраняет эмбединги и фрагменты в векторном хранилище Chroma DB.

26.48 OpenAI

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **OpenAI**.

Для получения дополнительной информации о функциях и возможностях OpenAI, используемых этими компонентами, см. документацию OpenAI.

Генерация текста OpenAI

Компонент **OpenAI** генерирует текст с использованием языковых моделей OpenAI.

Он предоставляет доступ к тем же моделям OpenAI, что и базовый компонент **Языковая модель**, но дополнительно позволяет настраивать параметры запроса к API OpenAI.

Компонент может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда нужно передать конкретную конфигурацию модели OpenAI в качестве LLM другому компоненту, например **Агент** или **Умное преобразование**.

Для получения дополнительной информации см. компоненты языковых моделей.

Параметры генерации текста OpenAI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Изменить все параметры компонента можно через панель инспекции компонента, которая появляется при его выборе.

Название	Тип	Описание
<code>api_key</code>	SecretString	Входной параметр. Ваш ключ API OpenAI.
<code>model</code>	String	Входной параметр. Имя используемой модели OpenAI. Доступные варианты включают gpt-3.5-turbo и gpt-4.

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Значение 0 означает отсутствие ограничений.
temperature	Float	Входной параметр. Управляет случайностью вывода. Диапазон: [0.0, 1.0]. По умолчанию: 0.7.
top_p	Float	Входной параметр. Управляет ядерной выборкой (nucleus sampling). Диапазон: [0.0, 1.0]. По умолчанию: 1.0.
frequency_penalty	Float	Входной параметр. Штраф за частоту повторения токенов. Диапазон: [0.0, 2.0]. По умолчанию: 0.0.
presence_penalty	Float	Входной параметр. Штраф за наличие уже встречавшихся токенов. Диапазон: [0.0, 2.0]. По умолчанию: 0.0.

Эмбединги OpenAI

Компонент **Эмбединги OpenAI** использует модели эмбедингов OpenAI для генерации векторных представлений текста.

Он предоставляет доступ к тем же моделям OpenAI, что и базовый компонент **Модель эмбедингов**, но дополнительно позволяет настраивать параметры запроса к API эмбедингов OpenAI.

Для получения дополнительной информации об использовании компонентов моделей эмбедингов в потоках см. компоненты моделей эмбедингов.

Параметры эмбедингов OpenAI

Некоторые параметры скрыты по умолчанию в визуальном редакторе. Изменить все параметры компонента можно через панель инспекции компонента, которая появляется при его выборе.

Название	Тип	Описание
Ключ API OpenAI	Строка	Входной параметр. Ключ API для доступа к API OpenAI.
Заголовки по умолчанию	Словарь	Входной параметр. Заголовки по умолчанию для HTTP-запросов.

Название	Тип	Описание
Параметры запроса по умолчанию	Вложенный словарь	Входной параметр. Параметры запроса по умолчанию для HTTP-запросов.
Allowed Special	Список	Входной параметр. Специальные токены, разрешённые для обработки. По умолчанию: [].
Disallowed Special	Список	Входной параметр. Специальные токены, запрещённые для обработки. По умолчанию: ["all"].
Размер фрагмента	Целое	Входной параметр. Размер блока для обработки. По умолчанию: 1000.
Клиент	Любой	Входной параметр. HTTP-клиент для выполнения запросов.
Развёртывание	Строка	Входной параметр. Имя развёртывания модели. По умолчанию: text-embedding-3-small.
Длина контекста эмбеддингов	Целое	Входной параметр. Длина контекста при формировании эмбеддингов. По умолчанию: 8191.
Максимальное число повторных попыток	Целое	Входной параметр. Максимальное количество повторных попыток при неудачных запросах. По умолчанию: 6.
Модель	Строка	Входной параметр. Название используемой модели. По умолчанию: text-embedding-3-small.
Параметры модели	Вложенный словарь	Входной параметр. Дополнительные именованные аргументы для модели.

Название	Тип	Описание
Базовый URL API OpenAI	Строка	Входной параметр. Базовый URL API OpenAI.
Тип API OpenAI	Строка	Входной параметр. Тип API OpenAI.
Версия API OpenAI	Строка	Входной параметр. Версия API OpenAI.
Организация OpenAI	Строка	Входной параметр. Организация, связанная с ключом API.
Прокси OpenAI	Строка	Входной параметр. Прокси-сервер для запросов.
Время ожидания запроса	Число с плавающей точкой	Входной параметр. Таймаут HTTP-запросов (в секундах).
Показывать индикатор выполнения	Булево	Входной параметр. Отображать ли индикатор прогресса при обработке. По умолчанию: false.
Пропускать пустые	Булево	Входной параметр. Пропускать ли пустые входные данные. По умолчанию: false.
TikToken включён	Булево	Входной параметр. Включать ли токенизатор TikToken. По умолчанию: true.
Название модели TikToken	Строка	Входной параметр. Имя модели TikToken для токенизации.

26.49 OpenRouter

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **OpenRouter**.

Подробнее о функциях и возможностях OpenRouter, используемых в компонентах пакета, см. в документации OpenRouter.

Генерация текста OpenRouter

Этот компонент генерирует текст с использованием унифицированного API OpenRouter для AI-моделей различных поставщиков.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда нужно задействовать модель OpenRouter в качестве LLM для другого компонента, управляемого языковой моделью, — например, компонента **Агент** или **Умное преобразование**.

Подробнее см. в разделе Компоненты языковой модели.

Параметры компонента «Генерация текста OpenRouter»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
api_key	SecretString	Входной параметр. Ключ API OpenRouter для аутентификации.
site_url	String	Входной параметр. URL вашего сайта для отображения в рейтингах OpenRouter.
app_name	String	Входной параметр. Название вашего приложения для отображения в рейтингах OpenRouter.
provider	String	Входной параметр. Поставщик AI-модели.
model_name	String	Входной параметр. Модель для генерации ответов.
temperature	Float	Входной параметр. Управляет случайностью генерации. Диапазон: [0.0, 2.0]. По умолчанию: 0.7.
max_tokens	Integer	Входной параметр. Максимальное количество токенов в ответе.

26.50 Perplexity

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Perplexity**.

Подробнее о функциях и возможностях Perplexity, используемых в компонентах пакета, см. в документации Perplexity.

Генерация текста Perplexity

Этот компонент генерирует текст с использованием языковых моделей Perplexity.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда нужно задействовать модель Perplexity в качестве LLM для другого компонента, управляемого языковой моделью, — например, компонента **Агент** или **Умное преобразование**.

Подробнее см. в разделе Компоненты языковой модели.

Параметры компонента «Генерация текста Perplexity»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
model_name	String	Входной параметр. Название модели Perplexity. Доступны различные модели Sonar и Llama.
max_tokens	Integer	Входной параметр. Максимальное количество токенов в ответе.
api_key	SecretString	Входной параметр. Ключ API Perplexity для аутентификации.
temperature	Float	Входной параметр. Управляет случайностью генерации. По умолчанию: 0.75.
top_p	Float	Входной параметр. Максимальная суммарная вероятность токенов при nucleus-сэмплинге.
n	Integer	Входной параметр. Количество вариантов ответа, генерируемых для каждого запроса.

26.51 pgvector

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **pgvector**.

Векторное хранилище pgvector

Компонент **PGVector** читает и записывает данные в векторные хранилища PostgreSQL, используя экземпляр PGVector.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как `vector_store`, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы `LangChain` не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике `API LangChain` и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов `Data` или табличного `DataFrame`. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Векторное хранилище `pgvector`»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации `pgvector` или в коде компонента.

Название	Тип	Описание
<code>pg_server_url</code>	<code>SecretString</code>	Входной параметр. Строка подключения к серверу PostgreSQL.
<code>collection_name</code>	<code>String</code>	Входной параметр. Имя таблицы для векторного хранилища.
<code>search_query</code>	<code>String</code>	Входной параметр. Запрос для поиска по схожести.
<code>ingest_data</code>	<code>Data</code>	Входной параметр. Данные для загрузки в векторное хранилище.
<code>embedding</code>	<code>Embeddings</code>	Входной параметр. Модель эмбедингов для использования.
<code>number_of_results</code>	<code>Integer</code>	Входной параметр. Количество результатов, возвращаемых при поиске.

26.52 Pinecone

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Pinecone**.

Векторное хранилище Pinecone

Компонент **Pinecone** читает и записывает данные в векторные хранилища Pinecone, используя экземпляр PineconeVectorStore.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как `vector_store`, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Векторное хранилище Pinecone»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Pinecone или в коде компонента.

Название	Тип	Описание
<code>index_name</code>	String	Входной параметр. Имя индекса Pinecone.
<code>namespace</code>	String	Входной параметр. Пространство имён для изоляции данных внутри индекса.

Название	Тип	Описание
distance_strategy	String	Входной параметр. Метрика расстояния для вычисления сходства векторов.
pinecone_api_key	SecretString	Входной параметр. Ключ API для Pinecone.
text_key	String	Входной параметр. Ключ в записи, используемый в качестве текста документа.
search_query	String	Входной параметр. Запрос для поиска по схожести.
ingest_data	Data	Входной параметр. Данные для загрузки в векторное хранилище.
embedding	Embeddings	Входной параметр. Модель эмбедингов для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.

26.53 Qdrant

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Qdrant**.

Векторное хранилище Qdrant

Компонент **Qdrant** читает и записывает данные в векторные хранилища Qdrant, используя экземпляр QdrantVectorStore.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как vector_store, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим

компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Векторное хранилище Qdrant»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Qdrant или в коде компонента.

Название	Тип	Описание
collection_name	String	Входной параметр. Имя коллекции Qdrant.
host	String	Входной параметр. Адрес хоста сервера Qdrant.
port	Integer	Входной параметр. Порт сервера Qdrant.
grpc_port	Integer	Входной параметр. gRPC-порт Qdrant.
api_key	SecretString	Входной параметр. Ключ API для Qdrant.
prefix	String	Входной параметр. Префикс URL для Qdrant.
timeout	Integer	Входной параметр. Тайм-аут операций Qdrant (в секундах).
path	String	Входной параметр. Путь к локальному хранилищу Qdrant.
url	String	Входной параметр. URL для подключения к Qdrant.
distance_func	String	Входной параметр. Метрика расстояния для вычисления сходства векторов.
content_payload_key	String	Входной параметр. Ключ payload, содержащий текст документа.

Название	Тип	Описание
metadata_payload_key	String	Входной параметр. Ключ payload, содержащий метаданные документа.
search_query	String	Входной параметр. Запрос для поиска по схожести.
ingest_data	Data	Входной параметр. Данные для загрузки в векторное хранилище.
embedding	Embeddings	Входной параметр. Модель эмбедингов для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.

26.54 Redis

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Redis**.

Память чата Redis

Компонент **Память чата Redis** извлекает и сохраняет сообщения чата, используя хранилище памяти Redis.

Память чата передаётся между компонентами хранилища памяти в виде типа данных Memory.

Подробнее об использовании внешней памяти чата в потоках см. компонент **История сообщений**.

Параметры компонента «Память чата Redis»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Отображаемое название	Описание
host	Хост	Входной параметр. IP-адрес или имя хоста.
port	Порт	Входной параметр. Номер порта Redis.
database	База данных	Входной параметр. Номер базы данных Redis.
username	Имя пользователя	Входной параметр. Имя пользователя Redis.
password	Пароль	Входной параметр. Пароль пользователя.

Название	Отображаемое название	Описание
key_prefix	Префикс ключа	Входной параметр. Префикс ключа для записей в Redis.
session_id	Идентификатор сеанса	Входной параметр. Уникальный идентификатор сессии для сообщений чата.

Векторное хранилище Redis

Компонент **векторного хранилища Redis** читает и записывает данные в векторные хранилища Redis, используя экземпляр RedisVectorStore.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как vector_store, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Векторное хранилище Redis»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Redis или в коде компонента.

Название	Тип	Описание
redis_server_url	SecretString	Входной параметр. Строка подключения к серверу Redis.

Название	Тип	Описание
redis_index_name	String	Входной параметр. Имя индекса Redis.
code	String	Входной параметр. Дополнительный пользовательский код для Redis.
schema	String	Входной параметр. Схема индекса Redis.
ingest_data	Data	Входной параметр. Данные для загрузки в векторное хранилище.
search_query	String	Входной параметр. Запрос для поиска по схожести.
embedding	Embeddings	Входной параметр. Модель эмбедингов для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.

26.55 SambaNova

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **SambaNova**.

Подробнее о функциях и возможностях SambaNova, используемых в компонентах пакета, см. в документации SambaNova Cloud.

Генерация текста SambaNova

Этот компонент генерирует текст с использованием LLM от SambaNova.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда нужно задействовать модель SambaNova в качестве LLM для другого компонента, управляемого языковой моделью, — например, компонента **Агент** или **Умное преобразование**.

Подробнее см. в разделе Компоненты языковой модели.

Параметры компонента «Генерация текста SambaNova»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
sambanova_url	String	Входной параметр. Базовый URL для API-запросов. По умолчанию:

Название	Тип	Описание
		https://api.sambanova.ai/v1/chat/completions .
sambanova_api_key	SecretString	Входной параметр. Ключ API SambaNova.
model_name	String	Входной параметр. Название модели SambaNova. Доступны различные модели Llama и другие.
max_tokens	Integer	Входной параметр. Максимальное количество токенов в ответе. Значение 0 означает отсутствие ограничения.
temperature	Float	Входной параметр. Управляет случайностью генерации. Диапазон: [0.0, 1.0]. По умолчанию: 0.07.

26.56 Serper

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Serper**.

Подробнее см. в документации Serper.

API Google Поиска Serper

Этот компонент позволяет вызывать API Google Поиска Serper.dev.

Он возвращает список результатов поиска в виде DataFrame.

Параметры API Google Поиска Serper

Название	Тип	Описание
Ключ API Serper (serper_api_key)	SecretString	Входной параметр. Ключ API для аутентификации в Serper.dev.
Поисковый запрос (input_value)	String	Входной параметр. Текст поискового запроса.
Количество результатов (k)	Integer	Входной параметр. Количество возвращаемых результатов поиска.

26.57 Supabase

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Supabase**.

Векторное хранилище Supabase

Компонент **Supabase** читает и записывает данные в векторные хранилища Supabase, используя экземпляр `SupabaseVectorStore`.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр `LangChain vector store` для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как `vector_store`, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы `LangChain` не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API `LangChain` и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов `Data` или табличного `DataFrame`. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Векторное хранилище Supabase»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Supabase или в коде компонента.

Название	Тип	Описание
<code>supabase_url</code>	<code>String</code>	Входной параметр. URL экземпляра Supabase.
<code>supabase_service_key</code>	<code>SecretString</code>	Входной параметр. Сервисный ключ для аутентификации в Supabase.
<code>table_name</code>	<code>String</code>	Входной параметр. Имя таблицы в Supabase.
<code>query_name</code>	<code>String</code>	Входной параметр. Имя SQL-функции поиска в Supabase.

Название	Тип	Описание
search_query	String	Входной параметр. Запрос для поиска по схожести.
ingest_data	Data	Входной параметр. Данные для загрузки в векторное хранилище.
embedding	Embeddings	Входной параметр. Модель эмбеддингов для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.

26.58 Upstash

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Upstash**.

Векторное хранилище Upstash

Компонент **Upstash** читает и записывает данные в векторные хранилища Upstash, используя экземпляр UpstashVectorStore.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для реализации операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как vector_store, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Векторное хранилище Upstash»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Upstash или в коде компонента.

Название	Тип	Описание
index_url	String	Входной параметр. URL индекса Upstash.
index_token	SecretString	Входной параметр. Токен для индекса Upstash.
text_key	String	Входной параметр. Ключ в записи, используемый как текст.
namespace	String	Входной параметр. Пространство имён для индекса.
search_query	String	Входной параметр. Запрос для поиска по схожести.
metadata_filter	String	Входной параметр. Фильтр документов по метаданным.
ingest_data	Data	Входной параметр. Данные для загрузки в векторное хранилище.
embedding	Embeddings	Входной параметр. Модель эмбеддингов для использования.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.

26.59 vLLM

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **vLLM**.

Подробнее о функциях и возможностях vLLM, используемых этими компонентами, см. в документации vLLM.

Генерация текста с помощью vLLM

Компонент **vLLM** генерирует текст с использованием моделей vLLM через совместимый с OpenAI API.

vLLM — это быстрая и простая в использовании библиотека для инференса и обслуживания LLM. Она обеспечивает высокую пропускную способность благодаря механизму PagedAttention, что делает её оптимальным выбором для самостоятельного развёртывания моделей.

Компонент подключается к серверу vLLM, работающему локально или удалённо, и использует совместимый с OpenAI API-эндпоинт для генерации текстовых ответов.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда хотите применять модель vLLM в качестве LLM для другого компонента, управляемого LLM, — например, компонента **Агент** или **Smart Function**.

Подробнее см. в разделе Компоненты языковой модели.

Параметры компонента «Генерация текста vLLM»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
api_key	SecretString	Входной параметр. API-ключ для модели vLLM (необязателен для локальных серверов).
model_name	String	Входной параметр. Имя модели vLLM для использования (например, ibm-granite/granite-3.3-8b-instruct).
api_base	String	Входной параметр. Базовый URL сервера vLLM. По умолчанию: http://localhost:8000/v1.
temperature	Float	Входной параметр. Управляет случайностью вывода. Диапазон: 0.0–1.0. По умолчанию: 0.1.
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для снятия ограничения.
seed	Integer	Входной параметр. Управляет воспроизводимостью результатов. По умолчанию: 1.
max_retries	Integer	Входной параметр. Максимальное количество повторных попыток при генерации. По умолчанию: 5.

Название	Тип	Описание
timeout	Integer	Входной параметр. Тайм-аут запросов к API vLLM (в секундах). По умолчанию: 700.
model_kwargs	Dict	Входной параметр. Дополнительные именованные аргументы, передаваемые модели.
json_mode	Boolean	Входной параметр. Если true, вывод всегда возвращается в формате JSON, независимо от передачи схемы.

26.60 Vectara

Пакеты содержат пользовательские компоненты, поддерживающие конкретные интеграции со сторонними сервисами в модуле «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Vectara**.

Векторное хранилище Vectara

Компонент **Vectara** читает и записывает данные в векторные хранилища Vectara, используя экземпляр векторного хранилища Vectara.

Об экземплярах векторного хранилища

Поскольку Генеративный ИИ основан на LangChain, компоненты векторного хранилища используют экземпляр LangChain vector store для реализации операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как `vector_store`, однако в некоторых компонентах векторного хранилища используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Параметры компонента «Векторное хранилище Vectara»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Vectara или в коде компонента.

Название	Тип	Описание
vectara_customer_id	String	Входной параметр. Идентификатор клиента Vectara.
vectara_corpus_id	String	Входной параметр. Идентификатор корпуса Vectara.
vectara_api_key	SecretString	Входной параметр. API-ключ Vectara.
embedding	Embeddings	Входной параметр. Модель эмбедингов для использования (необязательно).
ingest_data	List[Document/Data]	Входной параметр. Данные для загрузки в векторное хранилище.
search_query	String	Входной параметр. Запрос для поиска по схожести.
number_of_results	Integer	Входной параметр. Количество результатов, возвращаемых при поиске.

Vectara RAG

Этот компонент предоставляет полный сквозной RAG-пайплайн Vectara с поддержкой переранжирования результатов.

Компонент использует векторное хранилище Vectara для векторного поиска и переранжирования, после чего возвращает строку **Ответ** в формате Message.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как `vector_store`, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

26.61 Vertex AI

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Vertex AI**.

Подробнее о функциях и возможностях Vertex AI, используемых этими компонентами, см. в документации Vertex AI.

Другие компоненты Google см. в разделе Пакет **Google**.

Генерация текста с помощью Vertex AI

Компонент **Vertex AI** генерирует текст с использованием моделей Google Vertex AI, включая серии Gemini 1.5, 2.0, 2.5 и 3.0. Последние модели серии Gemini 3.0 (gemini-3-pro-preview, gemini-3-flash-preview и gemini-3-pro-image-preview) предлагают расширенные возможности рассуждения и мультимодальные функции.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда хотите применять модель Vertex AI в качестве LLM для другого компонента, управляемого LLM, — например, компонента **Агент** или **Умное преобразование**.

Подробнее см. в разделе Компоненты языковой модели.

Параметры компонента «Генерация текста Vertex AI»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
credentials	File	Входной параметр. JSON-файл учётных данных сервисного аккаунта. Оставьте пустым, чтобы использовать переменные окружения. Тип файла: JSON. Подробнее см. в разделе Создание сервисного аккаунта для аутентификации Vertex AI.
model_name	String	Входной параметр. Имя модели Vertex AI, например gemini-1.5-pro или gemini-3-pro-preview. По умолчанию: gemini-1.5-pro.
project	String	Входной параметр. Идентификатор проекта Google Cloud.
location	String	Входной параметр. Регион API Vertex AI. По умолчанию: us-central1.

Название	Тип	Описание
max_output_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации.
max_retries	Integer	Входной параметр. Максимальное количество повторных попыток при вызовах API. По умолчанию: 1.
temperature	Float	Входной параметр. Управляет случайностью вывода. По умолчанию: 0.0.
top_k	Integer	Входной параметр. Количество токенов с наивысшей вероятностью, сохраняемых при top-k-фильтрации.
top_p	Float	Входной параметр. Кумулятивная вероятность токенов с наивысшей вероятностью, сохраняемых при ядерной выборке. По умолчанию: 0.95.
verbose	Boolean	Входной параметр. Определяет, следует ли выводить подробные сведения о выполнении. По умолчанию: false.

Подробнее о параметрах генерации текста в Vertex AI см. в документации по параметрам генерации контента Vertex AI.

Эмбединги Vertex AI

Компонент **Эмбединги Vertex AI** представляет собой обёртку вокруг API эмбедингов Google Vertex AI для генерации векторных представлений текста.

Подробнее об использовании компонентов моделей эмбедингов в потоках см. в разделе Компоненты моделей эмбедингов.

Параметры компонента «Эмбединги Vertex AI»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
credentials	File	Входной параметр. JSON-файл учётных данных сервисного аккаунта. Оставьте пустым, чтобы использовать переменные

Название	Тип	Описание
		окружения. Тип файла: JSON. Подробнее см. в разделе Создание сервисного аккаунта для аутентификации Vertex AI.
location	String	Входной параметр. Регион по умолчанию для API-запросов. По умолчанию: us-central1.
max_output_tokens	Integer	Входной параметр. Максимальное количество токенов в ответе на один запрос. По умолчанию: 128.
model_name	String	Входной параметр. Имя модели Vertex AI. По умолчанию: text-bison.
project	String	Входной параметр. Идентификатор проекта GCP по умолчанию для запросов к Vertex API.
request_parallelism	Integer	Входной параметр. Степень параллелизма запросов к моделям Vertex AI. По умолчанию: 5.
temperature	Float	Входной параметр. Управляет степенью случайности при генерации. Должно быть неотрицательным значением. По умолчанию: 0.
top_k	Integer	Входной параметр. Задаёт число наиболее вероятных токенов, из которых выбирается следующий. По умолчанию: 40.
top_p	Float	Входной параметр. Токены выбираются от наиболее вероятных к наименее вероятным, пока сумма их вероятностей не превысит значение top_p. По умолчанию: 0.95.

Название	Тип	Описание
tuned_model_name	String	Входной параметр. Имя дообученной модели. Если указано, параметр model_name игнорируется.
verbose	Boolean	Входной параметр. При значении true выводятся внутренние состояния цепочки для отладки. По умолчанию: false.
embeddings	Embeddings	Выходной параметр. Экземпляр для генерации эмбеддингов с использованием Vertex AI.

26.62 Weaviate

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Weaviate**.

Векторное хранилище Weaviate

Компонент **Weaviate** читает и записывает данные в векторные хранилища Weaviate, используя экземпляр векторного хранилища Weaviate.

Об экземплярах векторных хранилищ

Поскольку Генеративный ИИ основан на LangChain, компоненты векторных хранилищ используют экземпляр LangChain vector store для выполнения базовых операций чтения и записи. Эти экземпляры зависят от поставщика и настраиваются в соответствии с параметрами компонента: строкой подключения, именем индекса и схемой.

В коде компонента он обычно создаётся как vector_store, однако в некоторых компонентах векторных хранилищ используется другое имя — как правило, имя поставщика.

Некоторые классы LangChain не предоставляют все возможные параметры в виде параметров компонента. В зависимости от поставщика такие параметры могут использовать значения по умолчанию или допускать настройку через переменные окружения, если они поддерживаются в модуле «Генеративный ИИ». Подробнее о конкретных параметрах см. в справочнике API LangChain и документации поставщика векторного хранилища.

Если вы используете компонент векторного хранилища для запроса к базе данных векторов, он возвращает результаты поиска, которые можно передать последующим компонентам в виде списка объектов Data или табличного DataFrame. Если поддерживаются оба типа, формат можно выбрать рядом с портом выхода компонента в визуальном редакторе.

Руководство по использованию базы данных векторов в потоке см. в разделе Создание чат-бота с векторным RAG.

Параметры компонента «Weaviate»

Параметры компонента векторного хранилища содержат подробную информацию о принимаемых входных данных, поддерживаемых функциях и способах их настройки.

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Некоторые параметры являются условными и становятся доступны только после задания других параметров или выбора определённых значений. Условные параметры могут не отображаться на панели **Управление** до установки необходимых зависимостей.

Допустимые значения и описание функциональности см. в документации Weaviate или в коде компонента.

Название	Тип	Описание
weaviate_url	String	Входной параметр. URL экземпляра Weaviate по умолчанию.
api_key	SecretString	Входной параметр. Необязательный API-ключ для аутентификации.
index_name	String	Входной параметр. Необязательное имя индекса.
text_key	String	Входной параметр. Ключ для извлечения текста по умолчанию.
input	Data or DataFrame	Входной параметр. Документ или запись.
cache_vector_store	Boolean	Входной параметр. Если true, векторное хранилище кэшируется в памяти для ускорения чтения. По умолчанию: включено (true).
embedding	Embeddings	Входной параметр. Подключите компонент модели встраивания.
number_of_results	Integer	Входной параметр. Количество возвращаемых результатов поиска. По умолчанию: 4.
search_by_text	Boolean	Входной параметр. Определяет, следует ли выполнять поиск по тексту. По умолчанию: отключено (false).

26.63 Wikipedia

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **Wikipedia**.

Компоненты Wikipedia полезны как инструменты для агентов, которым нужны надёжные источники информации, — например, обучающие чат-боты или универсальные исследовательские помощники.

Вместо широкого поиска по интернету или извлечения данных из внутренней модели эти компоненты предоставляют конкретные данные, полученные через Wikipedia API.

API Викиданных

Этот компонент выполняет поиск по схожести с использованием API Викиданных.

Он принимает текстовый поисковый запрос и возвращает DataFrame с результатами поиска.

Wikipedia API

Этот компонент ищет и извлекает информацию из Wikipedia с помощью поисковых эндпоинтов WikiMedia API и возвращает ответ в виде DataFrame.

Параметры компонента «Wikipedia API»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
input_value	String	Входной параметр. Текст поискового запроса.
lang	String	Входной параметр. Код языка для Wikipedia. По умолчанию: en.
k	Integer	Входной параметр. Количество возвращаемых результатов.
load_all_available_meta	Boolean	Входной параметр. Определяет, следует ли загружать все доступные метаданные.
doc_content_chars_max	Integer	Входной параметр. Максимальное количество символов в содержимом документа.

26.64 xAI

Пакеты содержат пользовательские компоненты, поддерживающие конкретные сторонние интеграции с модулем «Генеративный ИИ».

Эта страница описывает компоненты, доступные в пакете **xAI**.

Подробнее о функциях и возможностях xAI, используемых этими компонентами, см. в документации xAI.

Генерация текста с помощью xAI

Компонент **xAI** генерирует текст с использованием моделей xAI, таких как Grok.

Он может возвращать либо **Ответ модели** (Message), либо **Языковую модель** (LanguageModel).

Используйте выход **Языковая модель**, когда хотите применять модель xAI в качестве LLM для другого компонента, управляемого LLM, — например, компонента **Агент** или **Умное преобразование**.

Подробнее см. в разделе Компоненты языковой модели.

Параметры компонента «Генерация текста xAI»

Некоторые параметры по умолчанию скрыты в визуальном редакторе. Все параметры компонента можно изменить через панель инспекции компонента, которая появляется при выборе компонента.

Название	Тип	Описание
max_tokens	Integer	Входной параметр. Максимальное количество токенов для генерации. Установите 0 для снятия ограничения. Диапазон: 0–128000.
model_kwargs	Dict	Входной параметр. Дополнительные именованные аргументы для модели.
json_mode	Boolean	Входной параметр. Если true, вывод всегда возвращается в формате JSON, независимо от передачи схемы.
model_name	String	Входной параметр. Модель xAI для использования. По умолчанию: grok-2-latest.
base_url	String	Входной параметр. Базовый URL для API-запросов. По умолчанию: https://api.x.ai/v1.
api_key	SecretString	Входной параметр. API-ключ xAI для аутентификации.
temperature	Float	Входной параметр. Управляет случайностью вывода. Диапазон: 0.0–2.0. По умолчанию: 0.1.
seed	Integer	Входной параметр. Управляет воспроизводимостью результатов.

27. Начало работы с API

Вы можете использовать API Генеративный ИИ для программного взаимодействия с модулем «Генеративный ИИ», например:

- Создавать и редактировать потоки, включая управление файлами потоков.
- Разрабатывать приложения, использующие ваши потоки.
- Разрабатывать пользовательские компоненты.
- Интегрировать Генеративный ИИ как зависимость более крупного приложения, кодовой базы или сервиса.
- Вносить вклад в общую кодовую базу Генеративный ИИ.

Чтобы просмотреть и протестировать все доступные конечные точки, обратитесь к спецификации OpenAPI API Генеративный ИИ по конечной точке /docs вашего развёртывания Генеративный ИИ, например <http://localhost:7860/docs>.

Для примера использования API Генеративный ИИ в скрипте см. быстрый старт Генеративный ИИ.

Быстрый старт демонстрирует, как получить автоматически сгенерированные фрагменты кода для ваших потоков, использовать скрипт для запуска потока и извлекать данные из ответа API Генеративный ИИ.

27.1 Формирование запросов к API Генеративный ИИ

Хотя отдельные параметры различаются в зависимости от конечной точки, все запросы к API Генеративный ИИ имеют общие элементы: URL, метод, параметры и аутентификация.

В качестве примера следующая команда curl вызывает конечную точку /v1/run и передаёт переопределение времени выполнения (tweaks) компоненту **Вывод чата** потока:

```
curl --request POST \
  --url "$ASTRA_SERVER_URL/api/v1/run/$FLOW_ID?stream=false" \
  --header "Content-Type: application/json" \
  --header "x-api-key: $ASTRA_API_KEY" \
  --data '{
    "input_value": "hello world!",
    "output_type": "chat",
    "input_type": "chat",
    "tweaks": {
      "ChatOutput-6zcZt": {
        "should_store_message": true
      }
    }
  }'
```

Базовый URL

По умолчанию локальные развёртывания предоставляют API Генеративный ИИ по адресу <http://localhost:7860/api>.

Развёртывания Генеративный ИИ, размещённые удалённо, доступны по домену, установленному хостинг-сервисом, например http://IP_OR_DNS/api или http://IP_OR_DNS:ASTRA_PORT/api.

Вы можете настроить порт Генеративный ИИ через переменную окружения `ASTRA_PORT`.

- <https://UUID.ngrok.app/api>
- http://IP_OR_DNS/api

- `http://IP_OR_DNS:ASTRA_PORT/api`

Аутентификация

В версиях Генеративный ИИ 1.5 и новее большинство конечных точек API требуют аутентификации с помощью API-ключа Генеративный ИИ в заголовке `x-api-key` или параметре запроса. Для получения дополнительной информации см. Ключи API и аутентификация.

Как и в случае с любым API, следуйте отраслевым лучшим практикам по хранению и использованию чувствительных учётных данных. Например, вы можете установить переменные окружения для ваших API-ключей, а затем использовать эти переменные в запросах к API.

Методы, пути и параметры

Запросы к API Генеративный ИИ используют различные методы, пути, параметры пути, параметры запроса и параметры тела. Конкретные требования зависят от вызываемой конечной точки.

Например, для создания потока передайте определение потока в формате JSON в POST `/v1/flows`. Затем, чтобы запустить поток, вызовите POST `/v1/run/$FLOW_ID` с необязательными параметрами запуска в теле запроса.

Версии API

API Генеративный ИИ обслуживает конечные точки `/v1` и `/v2`.

Некоторые конечные точки существуют только в одной версии, а некоторые — в обеих.

Если запрос завершился ошибкой или дал неожиданный результат, убедитесь, что путь к конечной точке содержит правильную версию.

27.2 Настройка переменных окружения

Вы можете хранить часто используемые значения в переменных окружения для упрощения повторного использования, ротации токенов и безопасного обращения с чувствительными данными.

Вы можете использовать любой удобный способ установки переменных окружения: `export`, `.env`, `.zshrc` или `.curlrc`. Затем используйте эти переменные в запросах к API.

Например:

```
# Использование переменных окружения в запросах к API
curl --request POST \
  --url "$ASTRA_SERVER_URL/api/v1/run/$FLOW_ID?stream=false" \
  --header "Content-Type: application/json" \
  --header "x-api-key: $ASTRA_API_KEY" \
  --data '{
    "input_value": "hello world!",
    "output_type": "chat",
    "input_type": "chat",
    "tweaks": {
      "ChatOutput-6zcZt": {
        "should_store_message": true
      }
    }
  }'
```

Часто используемые значения в запросах к API Генеративный ИИ: URL сервера Генеративный ИИ, API-ключи Генеративный ИИ, идентификаторы потоков и идентификаторы проектов.

Идентификаторы потоков можно получить из панели **доступ к API**, из URL потока и с помощью GET /flows.

27.3 Попробуйте запросы к API Генеративный ИИ

Как только у вас будет URL сервера Генеративный ИИ, попробуйте вызвать следующие конечные точки, которые возвращают метаданные Генеративный ИИ.

Проверка работоспособности

Возвращает статус работоспособности базы данных и чат-сервиса Генеративный ИИ:

```
curl -X GET \
"$ASTRA_SERVER_URL/health_check" \
-H "accept: application/json"
```

Результат

```
{
  "status": "ok",
  "chat": "ok",
  "db": "ok"
}
```

Генеративный ИИ также предоставляет конечную точку GET /health. Эта конечная точка обслуживается uvicorn до полной инициализации Генеративный ИИ, поэтому она не является надёжной для проверки состояния сервиса Генеративный ИИ.

Получить версию

Возвращает текущую версию API Генеративный ИИ:

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/version" \
-H "accept: application/json"
```

Результат

```
{
  "version": "1.6.0",
  "main_version": "1.6.0",
  "package": "Генеративный ИИ"
}
```

Получить конфигурацию

Возвращает детали конфигурации вашего развёртывания Генеративный ИИ. Требуется API-ключ Генеративный ИИ.

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/config" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "feature_flags": {
    "mvp_components": false
  },
  "serialization_max_items_length": 1000,
  "serialization_max_text_length": 6000,
  "frontend_timeout": 0,
  "auto_saving": true,
  "auto_saving_interval": 1000,
  "health_check_max_retries": 5,
  "max_file_size_upload": 1024,
  "webhook_polling_interval": 5000,
  "public_flow_cleanup_interval": 3600,
  "public_flow_expiration": 86400,
  "event_delivery": "streaming",
  "webhook_auth_enable": false,
}
```

```
"voice_mode_available": false,  
"default_folder_name": "Starter Project",  
"hide_getting_started_progress": false  
}
```

Получить все компоненты

Возвращает словарь всех компонентов Генеративный ИИ. Требуется API-ключ Генеративный ИИ.

```
curl -X GET \  
"$ASTRA_SERVER_URL/api/v1/all" \  
-H "accept: application/json" \  
-H "x-api-key: $ASTRA_API_KEY"
```

27.4 Доступные конечные точки

Поскольку Генеративный ИИ можно запускать как IDE (фронтенд и бэкенд) или как среду выполнения (без UI, только бэкенд), он предоставляет конечные точки, поддерживающие операции как фронтенда, так и бэкенда. Многие конечные точки предназначены для оркестрации между фронтендом и бэкендом, чтения и записи в базу данных Генеративный ИИ, либо для включения функций фронтенда, таких как **Playground**. Если вы не вносите изменения в кодовую базу Генеративный ИИ, вам не потребуется напрямую вызывать большинство конечных точек.

Для разработки приложений наиболее часто используются конечные точки `/run` и `/webhook` для запуска потоков. В некоторых сценариях могут потребоваться другие конечные точки, например `/files` для работы с файлами в потоках.

Чтобы помочь вам изучить доступные конечные точки, ниже приведён список, отсортированный по основному сценарию использования (некоторые конечные точки могут поддерживать несколько сценариев).

Следующие конечные точки полезны для разработки приложений с модулем «Генеративный ИИ» и администрирования развёртываний с одним или несколькими пользователями. Чаще всего вы будете использовать конечные точки запуска потоков. Другие конечные точки пригодятся для специфических задач: администрирования и управления потоками в развёртываниях без визуального редактора.

- Конечные точки запуска потока:
 - `POST /v1/run/{flow_id_or_name}`: Запуск потока.
 - `POST /v1/run/advanced/{flow_id}`: Расширенный запуск с явным указанием `inputs`, `outputs`, `tweaks` и необязательного `session_id`.
 - `POST /v1/webhook/{flow_id_or_name}`: Запуск потока через `payload` вебхука.
- API ответов OpenAI:
 - `POST /v1/responses`: Выполнение потоков с использованием формата запроса, совместимого с OpenAI.
- Детали развёртывания:
 - `GET /v1/version`: Вернуть версию Генеративный ИИ. См. Получить версию.
 - `GET /v1/config`: Вернуть конфигурацию развёртывания. См. Получить конфигурацию.
 - `GET /health_check`: Конечная точка проверки работоспособности, валидирующая подключение к базе данных и чат-сервису. Возвращает статус 500, если какой-либо сервис недоступен.
- Конечные точки проектов:

- POST /v1/projects/: Создать проект.
- GET /v1/projects/: Список проектов.
- GET /v1/projects/{project_id}: Получить проект (с поддержкой постраничных потоков).
- PATCH /v1/projects/{project_id}: Обновить информацию о проекте и его участниках.
- DELETE /v1/projects/{project_id}: Удалить проект.
- GET /v1/projects/download/{project_id}: Экспортировать все потоки проекта в ZIP.
- POST /v1/projects/upload/: Импортировать ZIP-архив проекта (создаёт проект и потоки).
- GET /v1/starter-projects/: Вернуть список шаблонов.
- Конечные точки файлов:
 - Файлы (v1)
 - POST /v1/files/upload/{flow_id}: Загрузить файл в конкретный поток.
 - GET /v1/files/download/{flow_id}/{file_name}: Скачать файл из потока.
 - GET /v1/files/images/{flow_id}/{file_name}: Потокковая передача изображения из потока.
 - GET /v1/files/profile_pictures/{folder_name}/{file_name}: Получить ресурс изображения профиля.
 - GET /v1/files/profile_pictures/list: Список доступных изображений профиля.
 - GET /v1/files/list/{flow_id}: Список файлов для потока.
 - DELETE /v1/files/delete/{flow_id}/{file_name}: Удалить файл из потока.
 - Файлы (v2)
 - POST /v2/files: Загрузить файл, принадлежащий текущему пользователю.
 - GET /v2/files: Список файлов, принадлежащих текущему пользователю.
 - DELETE /v2/files/batch/: Удалить несколько файлов по их ID.
 - POST /v2/files/batch/: Скачать несколько файлов в виде ZIP по их ID.
 - GET /v2/files/{file_id}: Скачать файл по ID (или вернуть его необработанное содержимое).
 - PUT /v2/files/{file_id}: Переименовать файл по ID.
 - DELETE /v2/files/{file_id}: Удалить файл по ID.
 - DELETE /v2/files: Удалить все файлы текущего пользователя.
- Ключи API и аутентификация:
 - GET /v1/api_key/: Список API-ключей текущего пользователя.
 - POST /v1/api_key/: Создать новый API-ключ.
 - DELETE /v1/api_key/{api_key_id}: Удалить API-ключ.
 - POST /v1/api_key/store: Сохранить зашифрованный Store API-ключ (устанавливается cookie).
- Конечные точки управления потоками:
 - POST /v1/flows/: Создать поток.
 - GET /v1/flows/: Список потоков (поддерживает пагинацию и фильтры).
 - GET /v1/flows/{flow_id}: Получить поток по ID.
 - GET /v1/flows/public_flow/{flow_id}: Получить публичный поток по ID.

- PATCH /v1/flows/{flow_id}: Обновить поток.
- DELETE /v1/flows/{flow_id}: Удалить поток.
- POST /v1/flows/batch/: Создать несколько потоков.
- POST /v1/flows/upload/: Импортировать потоки из JSON-файла.
- DELETE /v1/flows/: Удалить несколько потоков по ID.
- POST /v1/flows/download/: Экспортировать потоки в ZIP-файл.
- GET /v1/flows/basic_examples/: Список базовых примерных потоков.
- Конечные точки пользователей:
 - POST /v1/users/: Добавить пользователя (требуется суперпользователь, если включена аутентификация).
 - GET /v1/users/whoami: Вернуть текущего аутентифицированного пользователя.
 - GET /v1/users/: Список всех пользователей (требуется суперпользователь).
 - PATCH /v1/users/{user_id}: Обновить пользователя (с проверкой ролей).
 - PATCH /v1/users/{user_id}/reset-password: Сбросить собственный пароль.
 - DELETE /v1/users/{user_id}: Удалить пользователя (нельзя удалить себя).
- Пользовательские компоненты:
 - GET /v1/all: Вернуть все доступные типы компонентов Генеративный ИИ. См. Получить все компоненты.
 - POST /v1/custom_component: Скомпилировать пользовательский компонент из кода и вернуть его узел.
 - POST /v1/custom_component/update: Обновить конфигурацию сборки и выводы существующего пользовательского компонента.
 - POST /v1/validate/code: Проверить фрагмент кода Python для пользовательского компонента.
 - POST /v1/validate/prompt: Проверить полезную нагрузку промпта.

Следующие конечные точки чаще всего используются при внесении вклада в кодовую базу Генеративный ИИ, так как они поддерживают оркестрацию фронтенда и бэкенда или другую внутреннюю функциональность.

- Основное (метаданные):
 - GET /v1/all: Вернуть все доступные типы компонентов Генеративный ИИ. См. Получить все компоненты.
 - GET /v1/version: Вернуть версию Генеративный ИИ. См. Получить версию.
 - GET /v1/config: Вернуть конфигурацию развёртывания. См. Получить конфигурацию.
 - GET /v1/starter-projects/: Вернуть список шаблонов.
- Конечные точки сборки (внутренняя поддержка редактора):
 - POST /v1/build/{flow_id}/flow: Запустить сборку потока и вернуть ID задания.
 - GET /v1/build/{job_id}/events: Поток передаточных данных или получение событий сборки.
 - POST /v1/build/{job_id}/cancel: Отменить задание сборки.
 - POST /v1/build_public_tmp/{flow_id}/flow: Сборка публичного потока без аутентификации.
 - POST /v1/validate/prompt: Проверить полезную нагрузку промпта.
- Ключи API и аутентификация:
 - POST /v1/login: Войти и установить токены в виде cookie.

- GET /v1/auto_login: Автоматический вход (если включено) и установка токенов.
 - POST /v1/refresh: Обновить токены с использованием refresh cookie.
 - POST /v1/logout: Выйти из системы и очистить cookie.
- Конечные точки мониторинга:
 - GET /v1/monitor/builds: Получить сборки вершин для потока.
 - DELETE /v1/monitor/builds: Удалить сборки вершин для потока.
 - GET /v1/monitor/messages/sessions: Список ID сессий сообщений (требуется аутентификация).
 - GET /v1/monitor/messages: Список сообщений с необязательными фильтрами.
 - DELETE /v1/monitor/messages: Удалить сообщения по ID (требуется аутентификация).
 - PUT /v1/monitor/messages/{message_id}: Обновить сообщение.
 - PATCH /v1/monitor/messages/session/{old_session_id}: Изменить ID сессии для всех сообщений в этой сессии.
 - DELETE /v1/monitor/messages/session/{session_id}: Удалить сообщения по сессии.
 - GET /v1/monitor/transactions: Список транзакций для потока (с пагинацией).
- Переменные:
 - POST /v1/variables/: Создать переменную (например, API-ключ) для пользователя.
 - GET /v1/variables/: Список переменных пользователя.
 - PATCH /v1/variables/{variable_id}: Обновить переменную.
 - DELETE /v1/variables/{variable_id}: Удалить переменную.
- Голосовой режим:
 - WS /v1/voice/ws/flow_as_tool/{flow_id}: Двухнаправленная голосовая сессия, предоставляющая поток в виде инструмента.
 - WS /v1/voice/ws/flow_as_tool/{flow_id}/{session_id}: То же, но с явным идентификатором сессии.
 - WS /v1/voice/ws/flow_tts/{flow_id}: Голос-в-текст сессия, запускающая поток и возвращающая TTS.
 - WS /v1/voice/ws/flow_tts/{flow_id}/{session_id}: То же, но с явным идентификатором сессии.
 - GET /v1/voice/elevenlabs/voice_ids: Список доступных идентификаторов голосов ElevenLabs для пользователя.
- MCP-серверы: Следующие конечные точки предназначены для управления MCP-серверами Генеративный ИИ и их соединениями. Обычно они не вызываются напрямую, а используются для внутренней функциональности в интерфейсе Генеративный ИИ. Серверы MCP поддерживают как потоковый HTTP, так и транспорт SSE.
 - HEAD /v1/mcp/streamable: Проверка работоспособности потокового HTTP MCP.
 - GET /v1/mcp/streamable: Открыть потоковое HTTP-соединение для сервера MCP.
 - POST /v1/mcp/streamable: Отправить сообщения на сервер MCP через потоковый HTTP.
 - DELETE /v1/mcp/streamable: Закрыть потоковое HTTP-соединение.

- HEAD /v1/mcp/sse (LEGACY): Проверка работоспособности MCP SSE.
- GET /v1/mcp/sse (LEGACY): Открыть SSE-поток для событий сервера MCP.
- POST /v1/mcp/ (LEGACY): Отправить сообщения на сервер MCP.
- GET /v1/mcp/project/{project_id}: Список инструментов с поддержкой MCP и настроек аутентификации проекта.
- HEAD /v1/mcp/project/{project_id}/streamable: Проверка работоспособности проектного потокового HTTP MCP.
- GET /v1/mcp/project/{project_id}/streamable: Открыть проектно-ориентированное потоковое HTTP-соединение.
- POST /v1/mcp/project/{project_id}/streamable: Отправить сообщения на проектный MCP-сервер через потоковый HTTP.
- DELETE /v1/mcp/project/{project_id}/streamable: Закрыть проектное потоковое HTTP-соединение.
- HEAD /v1/mcp/project/{project_id}/sse (LEGACY): Проверка работоспособности проектного SSE.
- GET /v1/mcp/project/{project_id}/sse (LEGACY): Открыть проектно-ориентированный MCP SSE.
- POST /v1/mcp/project/{project_id} (LEGACY): Отправить сообщения на проектный MCP-сервер.
- PATCH /v1/mcp/project/{project_id}: Обновить настройки MCP для потоков и аутентификации проекта.
- POST /v1/mcp/project/{project_id}/install: Установить конфигурацию клиента MCP для Cursor/Windsurf/Claude (только локально).
- GET /v1/mcp/project/{project_id}/installed: Проверить, у каких клиентов установлена конфигурация MCP.
- Пользовательские компоненты:
 - GET /v1/all: Вернуть все доступные типы компонентов Генеративный ИИ. См. Получить все компоненты.
 - POST /v1/custom_component: Создать пользовательский компонент из кода и вернуть его узел.
 - POST /v1/custom_component/update: Обновить конфигурацию сборки и выходные данные существующего пользовательского компонента.
 - POST /v1/validate/code: Проверить фрагмент кода Python для пользовательского компонента.
 - POST /v1/validate/prompt: Проверить полезную нагрузку промпта.

Следующие конечные точки устарели:

- POST /v1/predict/{flow_id}: Используйте /v1/run/{flow_id} вместо этого.
- POST /v1/process/{flow_id}: Используйте /v1/run/{flow_id} вместо этого.
- GET /v1/task/{task_id}: Устаревшая функциональность.
- POST /v1/upload/{flow_id}: Используйте /files вместо этого.
- POST /v1/build/{flow_id}/vertices: Заменено на /monitor/builds.
- POST /v1/build/{flow_id}/vertices/{vertex_id}: Заменено на /monitor/builds.
- GET /v1/build/{flow_id}/{vertex_id}/stream: Заменено на /monitor/builds.
- GET /v1/store/check/: Возвращает, включена ли функция Store.

- GET /v1/store/check/api_key: Проверяет, существует ли Store API-ключ и действителен ли он.
- POST /v1/store/components/: Опубликовать компонент в Store.
- PATCH /v1/store/components/{component_id}: Обновить опубликованный компонент.
- GET /v1/store/components/: Список доступных компонентов Store (поддерживаются фильтры).
- GET /v1/store/components/{component_id}: Скачать компонент из Store.
- GET /v1/store/tags: Список тегов Store.
- GET /v1/store/users/likes: Список компонентов, понравившихся текущему пользователю.
- POST /v1/store/users/likes/{component_id}: Поставить лайк компоненту.

28. TypeScript клиент

Клиент Генеративный ИИ для TypeScript позволяет TypeScript-приложениям программно взаимодействовать с API Генеративный ИИ.

Репозиторий клиента: `astra-multiagent-ai-client-ts`

Пакет npm: `@astra/multiagent-ai-client`

28.1 Установка пакета

Чтобы установить клиент Генеративный ИИ для TypeScript, используйте одну из следующих команд:

```
npm install @astra/multiagent-ai-client
```

```
yarn add @astra/multiagent-ai-client
```

```
pnpm add @astra/multiagent-ai-client
```

28.2 Инициализация клиента

1. Импортируйте клиент в ваш код:

```
import { AstraMultiagentAIClient } from "@astra/multiagent-ai-client";
```

2. Инициализируйте объект `AstraMultiagentAIClient` для взаимодействия с сервером:

```
const baseUrl = "BASE_URL";
```

```
const apiKey = "API_KEY";
```

```
const client = new AstraMultiagentAIClient({ baseUrl, apiKey });
```

Замените `BASE_URL` и `API_KEY` значениями из вашего развёртывания. Базовый URL Генеративный ИИ по умолчанию: `http://localhost:7860`. Чтобы создать API-ключ, см. API-ключи и аутентификация.

28.3 Подключение к серверу и получение ответов

1. После инициализации клиента проверьте соединение, вызвав сервер Генеративный ИИ. В следующем примере запускается поток (`runFlow`) путём передачи идентификатора потока и строки ввода чата:

```
import { AstraMultiagentAIClient } from "@astra/multiagent-ai-client";
```

```
const baseUrl = "http://localhost:7860";
```

```
const client = new AstraMultiagentAIClient({ baseUrl });
```

```
async function runFlow() {
```

```
  const flowId = "aa5a238b-02c0-4f03-bc5c-cc3a83335cdf";
```

```
  const flow = client.flow(flowId);
```

```
  const input = "Is anyone there?";
```

```
  const response = await flow.run(input);
```

```
  console.log(response);
```

```
}
```

```
runFlow().catch(console.error);
```

Замените следующие значения:

- `baseUrl`: URL вашего сервера Генеративный ИИ.
- `flowId`: идентификатор потока, который вы хотите запустить.
- `input`: сообщение чата, которое вы хотите отправить для запуска потока. Это применимо только к потокам с компонентом **Ввод в чат**.

2. Проверьте результат, чтобы убедиться, что клиент подключился к серверу Генеративный ИИ.

В следующем примере показан ответ на корректный запрос `runFlow`, который достиг сервера Генеративный ИИ и успешно запустил поток:

```
FlowResponse {  
  sessionId: 'aa5a238b-02c0-4f03-bc5c-cc3a83335cdf',  
  outputs: [ { inputs: [Object], outputs: [Array] } ]  
}
```

Ответ включает `session_id` — уникальный идентификатор клиент-серверного сеанса — и массив `outputs` с информацией о выполнении потока.

3. Необязательно: если вы хотите получить полные объекты ответа от сервера, измените `console.log`, чтобы преобразовать возвращаемый JSON-объект в строку:

```
console.log(JSON.stringify(response, null, 2));
```

Точная структура объектов `inputs` и `outputs` зависит от компонентов и конфигурации потока.

4. Необязательно: если вы хотите, чтобы ответ включал только сообщение чата из компонента **Вывод чата**, используйте удобную функцию `chatOutputText`:

```
console.log(response.chatOutputText());
```

28.4 Расширенные возможности клиента

Клиент TypeScript умеет не только подключаться к серверу и запускать поток.

Этот пример основан на быстром старте и добавляет дополнительные возможности для взаимодействия с модулем «Генеративный ИИ»:

1. Передайте настройки в виде объекта вместе с запросом.

Настройки (`tweaks`) — это программные переопределения параметров компонентов во время выполнения.

В этом примере меняется LLM, используемая компонентом языковой модели в потоке:

```
const tweaks = { model_name: "gpt-4o-mini" };
```

2. Передайте идентификатор сеанса вместе с запросом, чтобы отделить разговор от других запусков потока и иметь возможность продолжить его, используя тот же идентификатор в будущем:

```
const session_id = "aa5a238b-02c0-4f03-bc5c-cc3a83335cdf";
```

3. Вместо вызова `run` у объекта `Flow` вызовите `stream` с теми же аргументами, чтобы получить потоковый ответ:

```
const response = await client.flow(flowId).stream(input);
```

```
for await (const event of response) {  
  console.log(event);  
}
```

Ответ представляет собой `ReadableStream` объектов. Для получения дополнительной информации о потоковых ответах Генеративный ИИ см. `/run endpoint`.

4. Запустите изменённое приложение TypeScript, чтобы выполнить поток с `tweaks`, `session_id` и потоковой передачей:

```
import { AstraMultiagentAIClient } from "@astra/multiagent-ai-client";
```

```
const baseUrl = "http://localhost:7860";
```

```
const client = new AstraMultiagentAIClient({ baseUrl });
```

```
async function runFlow() {
```

```
  const flowId = "aa5a238b-02c0-4f03-bc5c-cc3a83335cdf";
```

```
  const input = "Is anyone there?";
```

```

const tweaks = { model_name: "gpt-4o-mini" };
const session_id = "test-session";

const response = await client.flow(flowId).stream(input, {
  session_id,
  tweaks,
});

for await (const event of response) {
  console.log(event);
}

runFlow().catch(console.error);

```

Замените следующие значения:

- baseUrl: URL вашего сервера Генеративный ИИ.
- flowId: идентификатор потока, который вы хотите запустить.
- input: сообщение чата, которое вы хотите отправить для запуска потока, если в потоке есть компонент **Ввод в чат**.
- tweaks: любые изменения настроек, применяемые к запуску потока. В этом примере меняется LLM, используемая в потоке.
- session_id: пользовательский идентификатор сеанса.

Результат

При включённой потоковой передаче ответ включает метаданные потока и события с отметками времени. Например:

```

{
  event: 'add_message',
  data: {
    timestamp: '2025-05-23 15:52:48 UTC',
    sender: 'User',
    sender_name: 'User',
    session_id: 'test-session',
    text: 'Is anyone there?',
    files: [],
    error: false,
    edit: false,
    properties: {
      text_color: "",
      background_color: "",
      edited: false,
      source: [Object],
      icon: "",
      allow_markdown: false,
      positive_feedback: null,
      state: 'complete',
      targets: []
    },
    category: 'message',
    content_blocks: [],
    id: '7f096715-3f2d-4d84-88d6-5e2f76bf3fbe',
    flow_id: 'aa5a238b-02c0-4f03-bc5c-cc3a83335cdf',
    duration: null
  }
}
{
  event: 'token',
  data: {
    chunk: 'Absolutely',
    id: 'c5a99314-6b23-488b-84e2-038aa3e87fb5',

```

```

    timestamp: '2025-05-23 15:52:48 UTC'
  }
}
{
  event: 'token',
  data: {
    chunk: ',',
    id: 'c5a99314-6b23-488b-84e2-038aa3e87fb5',
    timestamp: '2025-05-23 15:52:48 UTC'
  }
}
{
  event: 'token',
  data: {
    chunk: " I'm",
    id: 'c5a99314-6b23-488b-84e2-038aa3e87fb5',
    timestamp: '2025-05-23 15:52:48 UTC'
  }
}
{
  event: 'token',
  data: {
    chunk: ' here',
    id: 'c5a99314-6b23-488b-84e2-038aa3e87fb5',
    timestamp: '2025-05-23 15:52:48 UTC'
  }
}
}

// this response is abbreviated

{
  event: 'end',
  data: { result: { session_id: 'test-session', outputs: [Array] } }
}

```

28.5 Получение журналов с помощью клиента TypeScript

Чтобы получить журналы Генеративный ИИ, необходимо включить их извлечение на сервере Генеративный ИИ, добавив следующие значения в файл .env:

```

ASTRA_MULTIAGENT_AI_ENABLE_LOG_RETRIEVAL=True
ASTRA_MULTIAGENT_AI_LOG_RETRIEVER_BUFFER_SIZE=10000
ASTRA_MULTIAGENT_AI_LOG_LEVEL=DEBUG

```

Следующий пример запускает потоковую передачу журналов в фоновом режиме, а затем запускает поток, чтобы вы могли наблюдать за его выполнением:

```

const baseUrl = "http://localhost:7863";
const flowId = "86f0bf45-0544-4e88-b0b1-8e622da7a7f0";

async function runFlow(client: AstraMultiagentAIClient) {
  const input = "Is anyone there?";
  const response = await client.flow(flowId).run(input);
  console.log("Flow response:", response);
}

async function main() {
  const client = new AstraMultiagentAIClient({ baseUrl });

  console.log("Starting log stream...");
  for await (const log of await client.logs.stream()) {
    console.log("Log:", log);
  }
}

```

```
await runFlow(client);
}
```

```
main().catch(console.error);
```

Замените следующие значения:

- baseUrl: URL вашего сервера Генеративный ИИ.
- flowId: идентификатор потока, который вы хотите запустить.
- input: сообщение чата, которое вы хотите отправить для запуска потока, если в потоке есть компонент **Ввод в чат**.

Журналы начинают передаваться в потоковом режиме без ограничений, а поток запускается один раз.

Результат

Следующий пример результата сокращён для удобства чтения, но по сообщениям можно проследить, как поток создаёт компоненты, настраивает модель и обрабатывает выводы.

Объект FlowResponse в конце потока возвращается клиенту вместе с результатом выполнения в массиве outputs.

Starting log stream...

```
Log: Log {
  timestamp: 2025-05-30T11:49:16.006Z,
  message: '2025-05-30T07:49:16.006127-0400 DEBUG Instantiating ChatInput of type component\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.029Z,
  message: '2025-05-30T07:49:16.029957-0400 DEBUG Instantiating Prompt of type component\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.049Z,
  message: '2025-05-30T07:49:16.049520-0400 DEBUG Instantiating ChatOutput of type component\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.069Z,
  message: '2025-05-30T07:49:16.069359-0400 DEBUG Instantiating OpenAIModel of type component\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.086Z,
  message: "2025-05-30T07:49:16.086426-0400 DEBUG Running layer 0 with 2 tasks, ['ChatInput-xjucM', 'Prompt-I3pxU']\n"
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.101Z,
  message: '2025-05-30T07:49:16.101766-0400 DEBUG Building Chat Input\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.113Z,
  message: '2025-05-30T07:49:16.113343-0400 DEBUG Building Prompt\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.131Z,
  message: '2025-05-30T07:49:16.131423-0400 DEBUG Logged vertex build: 6bd9fe9c-5eea-4f05-a96d-f6de9dc77e3c\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.143Z,
  message: '2025-05-30T07:49:16.143295-0400 DEBUG Logged vertex build: 39c68ec9-3859-4fff-9b14-80b3271f8fbf\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.188Z,
  message: "2025-05-30T07:49:16.188730-0400 DEBUG Running layer 1 with 1 tasks, ['OpenAIModel-RtlZm']\n"
```

```
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.201Z,
  message: '2025-05-30T07:49:16.201946-0400 DEBUG Building OpenAI\n'
}
Log: Log {
  timestamp: 2025-05-30T11:49:16.216Z,
  message: '2025-05-30T07:49:16.216622-0400 INFO Model name: gpt-4.1-mini\n'
}
Flow response: FlowResponse {
  sessionId: '86f0bf45-0544-4e88-b0b1-8e622da7a7f0',
  outputs: [ { inputs: [Object], outputs: [Array] } ]
}
Log: Log {
  timestamp: 2025-05-30T11:49:18.094Z,
  message: '2025-05-30T07:49:18.094364-0400 DEBUG Vertex OpenAIModel-RtlZm, result: , object: {\text_output':
  "Hey there! I'm here and ready to help you build something awesome with AI. What are you thinking about creating
  today?"}\n'
}
```

Для получения дополнительной информации см. конечные точки журналов.

29. Триггеры потока

Используйте конечные точки /run и /webhook для запуска потоков.

Для создания, чтения, обновления и удаления потоков см. Конечные точки управления потоками.

29.1 Запуск потока

Генеративный ИИ автоматически генерирует фрагменты кода на Python, JavaScript и curl для конечной точки /v1/run/\$FLOW_ID для всех потоков. Для получения дополнительной информации см. Создание фрагментов кода API. Выполните указанный поток по ID или имени. Идентификаторы потоков можно найти в фрагментах кода на панели **API доступ** или в URL потока.

В следующем примере запускается шаблонный поток **Основные подсказки** с параметрами потока, переданными в теле запроса. Этот поток требует строку ввода чата (input_value) и использует значения по умолчанию для всех остальных параметров.

```
url = "http://SERVER_URL/api/v1/run/FLOW_ID"

# Request payload
payload = {
  "input_value": "Tell me about something interesting!",
  "session_id": "chat-123",
  "input_type": "chat",
  "output_type": "chat",
  "output_component": ""
}

# Request headers
headers = {
  "Content-Type": "application/json",
  "x-api-key": "ASTRA_API_KEY"
}

try:
  response = requests.post(url, json=payload, headers=headers)
  response.raise_for_status()
  print(response.json())
except requests.exceptions.RequestException as e:
  print(f"Error making API request: {e}")

const payload = {
  input_value: "Tell me about something interesting!",
  session_id: "chat-123",
  input_type: "chat",
  output_type: "chat",
  output_component: ""
};

const options = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'x-api-key': 'ASTRA_API_KEY'
  },
  body: JSON.stringify(payload)
};
```

```

fetch('http://SERVER_URL/api/v1/run/FLOW_ID', options)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/run/$FLOW_ID" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "input_value": "Tell me about something interesting!",
  "session_id": "chat-123",
  "input_type": "chat",
  "output_type": "chat",
  "output_component": ""
}'

```

Ответ от /v1/run/\$FLOW_ID включает метаданные, входные данные и выходные данные выполнения.

Результат

В следующем примере показан ответ от потока Basic Prompting:

```

{
  "session_id": "chat-123",
  "outputs": [{
    "inputs": {
      "input_value": "Tell me about something interesting!"
    },
    "outputs": [{
      "results": {
        "message": {
          "text": "Sure! Have you ever heard of the phenomenon known as \"bioluminescence\"? It's a fascinating natural occurrence where living organisms produce and emit light. This ability is found in various species, including certain types of jellyfish, fireflies, and deep-sea creatures like anglerfish.\n\nBioluminescence occurs through a chemical reaction in which a light-emitting molecule called luciferin reacts with oxygen, catalyzed by an enzyme called luciferase. The result is a beautiful glow that can serve various purposes, such as attracting mates, deterring predators, or luring prey.\n\nOne of the most stunning displays of bioluminescence can be seen in the ocean, where certain plankton emit light when disturbed, creating a mesmerizing blue glow in the water. This phenomenon is often referred to as \"sea sparkle\" and can be seen in coastal areas around the world.\n\nBioluminescence not only captivates our imagination but also has practical applications in science and medicine, including the development of biosensors and imaging techniques. It's a remarkable example of nature's creativity and complexity!",
          "sender": "Machine",
          "sender_name": "AI",
          "session_id": "chat-123",
          "timestamp": "2025-03-03T17:17:37+00:00",
          "flow_id": "d2bbd92b-187e-4c84-b2d4-5df365704201",
          "properties": {
            "source": {
              "id": "OpenAIModel-d1wOZ",
              "display_name": "OpenAI",
              "source": "gpt-4o-mini"
            },
            "icon": "OpenAI"
          },
          "component_id": "ChatOutput-ylMzN"
        }
      }
    ]
  }
}

```

Если вы разбираете ответ в приложении, вам, скорее всего, потребуется извлечь релевантный контент из ответа, а не передавать весь ответ пользователю. Для примера скрипта, извлекающего данные из ответа API Генеративный ИИ, см. Быстрый старт.

Потоковая передача ответы токенов LLM

С помощью `/v1/run/$FLOW_ID` поток выполняется в виде пакета с опциональной потоковой передачей ответов токенов LLM.

Чтобы потоково передавать ответы токенов LLM, добавьте параметр запроса `?stream=true` к запросу:

```
url = "http://SERVER_URL/api/v1/run/FLOW_ID?stream=true"

# Request payload
payload = {
  "message": "Tell me something interesting!",
  "session_id": "chat-123"
}

# Request headers
headers = {
  "accept": "application/json",
  "Content-Type": "application/json",
  "x-api-key": "ASTRA_API_KEY"
}

try:
  response = requests.post(url, json=payload, headers=headers, stream=True)
  response.raise_for_status()

  # Process streaming response
  for line in response.iter_lines():
    if line:
      print(line.decode('utf-8'))
except requests.exceptions.RequestException as e:
  print(f"Error making API request: {e}")

const payload = {
  message: "Tell me something interesting!",
  session_id: "chat-123"
};

const options = {
  method: 'POST',
  headers: {
    'accept': 'application/json',
    'Content-Type': 'application/json',
    'x-api-key': 'ASTRA_API_KEY'
  },
  body: JSON.stringify(payload)
};

fetch('http://SERVER_URL/api/v1/run/FLOW_ID?stream=true', options)
  .then(async response => {
    const reader = response.body?.getReader();
    const decoder = new TextDecoder();

    if (reader) {
      while (true) {
        const { done, value } = await reader.read();
        if (done) break;
        console.log(decoder.decode(value));
      }
    }
  })
  .catch(err => console.error(err));
```

```
curl -X POST \
"$ASTRA_SERVER_URL_URL/api/v1/run/$FLOW_ID?stream=true" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "message": "Tell me something interesting!",
  "session_id": "chat-123"
}'
```

Ответы чата LLM передаются в виде потоковых событий token, завершаясь финальным событием end, которое закрывает соединение.

Результат

Следующий пример усечён, чтобы проиллюстрировать серию событий token и финальное событие end, закрывающее потоковый ответ токенов LLM:

```
{
  "event": "add_message",
  "data": {
    "timestamp": "2025-03-03T17:20:18",
    "sender": "User",
    "sender_name": "User",
    "session_id": "chat-123",
    "text": "Tell me about something interesting!",
    "files": [],
    "error": false,
    "edit": false,
    "properties": {
      "text_color": "",
      "background_color": "",
      "edited": false,
      "source": {
        "id": null,
        "display_name": null,
        "source": null
      },
      "icon": "",
      "allow_markdown": false,
      "positive_feedback": null,
      "state": "complete",
      "targets": []
    },
    "category": "message",
    "content_blocks": [],
    "id": "0103a21b-cbf7-4c02-9d72-017fb297f812",
    "flow_id": "d2bbd92b-187e-4c84-b2d4-5df365704201"
  }
}

{
  "event": "add_message",
  "data": {
    "timestamp": "2025-03-03T17:20:18",
    "sender": "Machine",
    "sender_name": "AI",
    "session_id": "chat-123",
    "text": "",
    "files": [],
    "error": false,
    "edit": false,
    "properties": {
      "text_color": "",
      "background_color": "",
      "edited": false,
      "source": {
        "id": "OpenAIModel-d1wOZ",
        "display_name": "OpenAI",
        "source": "gpt-4o-mini"
      },
      "icon": "OpenAI",
      "allow_markdown": false,
      "positive_feedback": null,
      "state": "complete",
      "targets": []
    },
    "category": "message",
    "content_blocks": [],
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "flow_id": "d2bbd92b-187e-4c84-b2d4-5df365704201"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " Have",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " you",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " ever",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " heard",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " of",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " the",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "token",
  "data": {
    "chunk": " phenomenon",
    "id": "27b66789-e673-4c65-9e81-021752925161",
    "timestamp": "2025-03-03 17:20:18 UTC"
  }
}

{
  "event": "end",
  "data": {
    "result": {
      "session_id": "chat-123",
      "message": "Sure! Have you ever heard of the phenomenon known as \"bioluminescence\"?..."
    }
  }
}
```

Заголовки конечной точки запуска

Заголовок	Информация	Пример
Content-Type	Обязательно.	“application/json”
accept	Указывает формат JSON. Необязательно. Указывает формат ответа.	“application/json”

Заголовок	Информация	Пример
x-api-key	По умолчанию JSON, если не указано. Обязательно. Ваш API-ключ Генеративный ИИ для аутентификации. Может быть передан в заголовке или как параметр запроса.	“sk-...”
X-ASTRA-GLOBAL-VAR-*	Необязательно. Передайте глобальные переменные в поток. Имена переменных автоматически преобразуются в верхний регистр. Эти переменные имеют приоритет над переменными окружения ОС и доступны только во время выполнения данного запроса.	"X-ASTRA-GLOBAL-VAR-API_KEY: sk-..."
Параметры конечной точки запуска		
Параметр	Тип	Информация
flow_id	UUID/string	Обязательно. Часть URL: /run/\$FLOW_ID
stream	Boolean	Необязательно. Параметр Запрос: /run/\$FLOW_ID?stream=true
input_value	string	Необязательно. Поле тела JSON. Основной ввод текста/подсказка. По умолчанию: null
input_type	string	Необязательно. Поле тела JSON. Тип ввода (“chat” или “text”). По умолчанию: "chat"
output_type	string	Необязательно. Поле тела JSON. Тип вывода (“chat”, “any”, “debug”). По умолчанию: "chat"
output_component	string	Необязательно. Поле тела JSON. Целевой компонент для вывода. По умолчанию: ""
tweaks	object	Необязательно. Поле тела JSON. Настройки

Параметр	Тип	Информация компонента.	По умолчанию: null
session_id	string	Необязательно. Поле тела JSON. Идентификатор контекста разговора. См. Идентификатор сеанса. По умолчанию: null	

Пример запроса со всеми заголовками и параметрами

```
url = "http://SERVER_URL/api/v1/run/FLOW_ID?stream=true"
```

```
# Request payload with tweaks
```

```
payload = {
    "input_value": "Tell me a story",
    "input_type": "chat",
    "output_type": "chat",
    "output_component": "chat_output",
    "session_id": "chat-123",
    "tweaks": {
        "component_id": {
            "parameter_name": "value"
        }
    }
}
```

```
# Request headers
```

```
headers = {
    "Content-Type": "application/json",
    "accept": "application/json",
    "x-api-key": "ASTRA_API_KEY"
}
```

```
try:
```

```
    response = requests.post(url, json=payload, headers=headers, stream=True)
    response.raise_for_status()
```

```
# Process streaming response
```

```
for line in response.iter_lines():
```

```
    if line:
```

```
        print(line.decode('utf-8'))
```

```
except requests.exceptions.RequestException as e:
```

```
    print(f"Error making API request: {e}")
```

```
const payload = {
```

```
    input_value: "Tell me a story",
```

```
    input_type: "chat",
```

```
    output_type: "chat",
```

```
    output_component: "chat_output",
```

```
    session_id: "chat-123",
```

```
    tweaks: {
```

```
        component_id: {
```

```
            parameter_name: "value"
```

```
        }
```

```
    }
```

```
};
```

```
const options = {
```

```
    method: 'POST',
```

```
    headers: {
```

```

'Content-Type': 'application/json',
'accept': 'application/json',
'x-api-key': 'ASTRA_API_KEY'
},
body: JSON.stringify(payload)
};

fetch('http://SERVER_URL/api/v1/run/FLOW_ID?stream=true', options)
.then(async response => {
  const reader = response.body?.getReader();
  const decoder = new TextDecoder();

  if (reader) {
    while (true) {
      const { done, value } = await reader.read();
      if (done) break;
      console.log(decoder.decode(value));
    }
  }
})
.catch(err => console.error(err));

curl -X POST \
"$ASTRA_SERVER_URL/api/v1/run/$FLOW_ID?stream=true" \
-H "Content-Type: application/json" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "input_value": "Tell me a story",
  "input_type": "chat",
  "output_type": "chat",
  "output_component": "chat_output",
  "session_id": "chat-123",
  "tweaks": {
    "component_id": {
      "parameter_name": "value"
    }
  }
}'

```

Передача глобальных переменных в заголовках запроса

Вы можете передать глобальные переменные в ваш поток, используя HTTP-заголовки в формате X-ASTRA-GLOBAL-VAR-{VARIABLE_NAME}.

Переменные, переданные в заголовках, имеют приоритет над переменными окружения ОС. Если переменная указана как в заголовке, так и в переменной окружения, используется значение из заголовка. Переменные доступны только во время выполнения конкретного запроса и не сохраняются.

Имена переменных автоматически преобразуются в верхний регистр. Например, X-ASTRA-GLOBAL-VAR-api-key становится API_KEY в вашем потоке.

Не требуется предварительно создавать эти переменные в разделе Глобальные переменные Генеративный ИИ. Передавайте любое имя переменной, используя формат заголовка.

```
url = "http://SERVER_URL/api/v1/run/FLOW_ID"
```

```

# Request payload
payload = {
  "input_value": "Tell me about something interesting!",
  "input_type": "chat",
  "output_type": "chat"
}

```

```

}

# Request headers with global variables
headers = {
  "Content-Type": "application/json",
  "x-api-key": "ASTRA_API_KEY",
  "X-ASTRA-GLOBAL-VAR-OPENAI_API_KEY": "sk-...",
  "X-ASTRA-GLOBAL-VAR-USER_ID": "user123",
  "X-ASTRA-GLOBAL-VAR-ENVIRONMENT": "production"
}

try:
  response = requests.post(url, json=payload, headers=headers)
  response.raise_for_status()
  print(response.json())
except requests.exceptions.RequestException as e:
  print(f"Error making API request: {e}")
const payload = {
  input_value: "Tell me about something interesting!",
  input_type: "chat",
  output_type: "chat"
};

const options = {
  method: 'POST',
  headers: {
    'Content-Type': 'application/json',
    'x-api-key': 'ASTRA_API_KEY',
    'X-ASTRA-GLOBAL-VAR-OPENAI_API_KEY': 'sk-...',
    'X-ASTRA-GLOBAL-VAR-USER_ID': 'user123',
    'X-ASTRA-GLOBAL-VAR-ENVIRONMENT': 'production'
  },
  body: JSON.stringify(payload)
};

fetch('http://SERVER_URL/api/v1/run/FLOW_ID', options)
  .then(response => response.json())
  .then(data => console.log(data))
  .catch(err => console.error(err));
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/run/$FLOW_ID" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-H "X-ASTRA-GLOBAL-VAR-OPENAI_API_KEY: sk-..." \
-H "X-ASTRA-GLOBAL-VAR-USER_ID: user123" \
-H "X-ASTRA-GLOBAL-VAR-ENVIRONMENT: production" \
-d '{
  "input_value": "Tell me about something interesting!",
  "input_type": "chat",
  "output_type": "chat"
}'

```

Если компоненты вашего потока ссылаются на переменные, которые не указаны в заголовках или в вашей базе данных Генеративный ИИ, поток по умолчанию завершится с ошибкой. Чтобы избежать этого, вы можете установить `ASTRA_FALLBACK_TO_ENV_VAR=True` в файле `.env`, что позволит потоку использовать значения из переменных окружения ОС, если они не заданы иначе.

29.2 Вебхук запуска потока

Используйте конечную точку `/webhook`, чтобы запустить поток, отправив HTTP POST запрос.

После того как вы добавите компонент **Вебхук** в поток, откройте панель **доступ к API** и нажмите вкладку **Вебхук curl**, чтобы получить автоматически сгенерированный запрос POST `/webhook` для вашего потока. Для получения дополнительной информации см. [Запуск потоков с помощью вебхуков](#).

```
curl -X POST \
"$ASTRA_SERVER_URL_URL/api/v1/webhook/$FLOW_ID" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{"data": "example-data"}
```

Результат

```
{
  "message": "Task started in the background",
  "status": "in progress"
}
```

29.3 Устаревшие конечные точки запуска потоков

Следующие конечные точки устарели и заменены конечной точкой `/run`:

- `/process`
- `/predict`

30. API разработчика (Beta)

30.1 Workflow API (Beta)

caution Бета-функция

Workflow API в настоящее время находится в статусе **Beta**. Конечные точки API и форматы ответов могут измениться в будущих версиях. Workflow API предоставляет программный доступ к выполнению рабочих процессов Генеративный ИИ в синхронном и асинхронном режимах. Синхронные запросы возвращают полный результат сразу после завершения. Асинхронные запросы ставятся в очередь и выполняются в фоновом режиме до завершения либо до отправки запроса к конечной точке остановки рабочего процесса.

Workflow API является частью Генеративный ИИ Developer API v2 и предоставляет более широкие возможности выполнения рабочих процессов по сравнению с конечной точкой v1 /run.

Требования

Перед использованием API необходимо:

- Установить и запустить Генеративный ИИ с включённым Developer API
Конечные точки Workflow API требуют включённой настройки `developer_api_enabled`. Если эта настройка отключена, эти конечные точки возвращают ошибку 404 Not Found.

Чтобы включить Developer API:

1. В файле `.env` задайте переменную окружения со значением `true`:
`ASTRA_MULTIAGENT_AI_DEVELOPER_API_ENABLED=true`
2. Запустите сервер Генеративный ИИ с использованием файла `.env`:
`uv run astra run --env-file .env`

Для получения дополнительной информации о настройке переменных окружения см. Переменные окружения.

- Создать API-ключ Генеративный ИИ
- Создать поток, который вы хотите выполнить
- Получить ID потока или имя конечной точки потока, который вы хотите выполнить

Настройка переменных окружения

Все примеры кода в этой документации предполагают, что вы задали следующие переменные окружения.

Python:

```
ASTRA_MULTIAGENT_AI_SERVER_URL = os.getenv("ASTRA_MULTIAGENT_AI_SERVER_URL")
ASTRA_MULTIAGENT_AI_API_KEY = os.getenv("ASTRA_MULTIAGENT_AI_API_KEY")
```

TypeScript/JavaScript:

```
const ASTRA_MULTIAGENT_AI_SERVER_URL = process.env.ASTRA_MULTIAGENT_AI_SERVER_URL;
const ASTRA_MULTIAGENT_AI_API_KEY = process.env.ASTRA_MULTIAGENT_AI_API_KEY;
```

Задайте эти переменные окружения перед запуском примеров или замените их в примерах кода на фактический URL сервера Генеративный ИИ и API-ключ.

По умолчанию `ASTRA_MULTIAGENT_AI_SERVER_URL` для локального развёртывания Генеративный ИИ имеет значение `http://localhost:7860`. Для удалённых развёртываний домен задаётся вашим хостинг-провайдером, например `https://UUID.ngrok.app`.

Аутентификация и заголовки

Все запросы Workflow API требуют аутентификации с использованием API-ключа Генеративный ИИ. API-ключ передаётся в заголовке `x-api-key`.

Для получения дополнительной информации см. Создание API-ключа Генеративный ИИ.

Заголовок	Описание	Пример
Content-Type	Указывает, что тело запроса передаётся в формате JSON.	application/json
x-api-key	API-ключ Генеративный ИИ.	sk-...
accept	Необязательно. Указывает ожидаемый формат ответа.	application/json

Конечная точка выполнения рабочих процессов

Конечная точка:

POST /api/v2/workflows

Описание: выполняет рабочий процесс синхронно или асинхронно. Установите background=false, чтобы выполнить запрос синхронно.

Пример синхронного запроса

Выполните рабочий процесс синхронно и получите полный результат сразу после завершения.

```
url = f'{ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows'
headers = {
    "Content-Type": "application/json",
    "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
}

payload = {
    "flow_id": "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
    "background": False,
    "inputs": {
        "ChatInput-abc.input_type": "chat",
        "ChatInput-abc.input_value": "what is 2+2",
        "ChatInput-abc.session_id": "session-123"
    }
}

response = requests.post(url, json=payload, headers=headers)
print(response.json())

const url = `${ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows`;

const payload = {
  flow_id: "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  background: false,
  inputs: {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "what is 2+2",
    "ChatInput-abc.session_id": "session-123"
  }
};

const runWorkflow = async () => {
  try {
    const response = await axios.post(url, payload, {
      headers: {
        "Content-Type": "application/json",
        "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
      }
    });
  } catch (error) {
    console.error(error);
  }
};
```

```

    }
  });
  console.log(response.data);
} catch (error) {
  console.error("Error triggering workflow:", error);
}
};

runWorkflow();
curl -X POST \
"$ASTRA_MULTIAGENT_AI_SERVER_URL/api/v2/workflows" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \
-d '{
  "flow_id": "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  "background": false,
  "inputs": {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "what is 2+2",
    "ChatInput-abc.session_id": "session-123"
  }
}'

```

Пример асинхронного запроса

Для длительных рабочих процессов установите `background=true`, чтобы сразу получить `job_id`, а затем периодически запрашивайте статус через конечную точку GET, пока задание не завершится.

Чтобы остановить задание, отправьте POST-запрос к конечной точке остановки рабочего процесса.

Асинхронный запрос содержит параметр `stream`, однако потоковая передача пока не поддерживается. Этот параметр включён для будущей совместимости. **Пример запроса:**

```

url = f"{ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows"
headers = {
  "Content-Type": "application/json",
  "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
}

payload = {
  "flow_id": "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  "background": True,
  "stream": False,
  "inputs": {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "Process this in the background",
    "ChatInput-abc.session_id": "session-456"
  }
}

response = requests.post(url, json=payload, headers=headers)
print(response.json()) # Returns job_id immediately

const url = `${ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows`;

const payload = {
  flow_id: "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  background: true,
  stream: false,
  inputs: {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "Process this in the background",

```

```
"ChatInput-abc.session_id": "session-456"
}
};

const runWorkflow = async () => {
  try {
    const response = await axios.post(url, payload, {
      headers: {
        "Content-Type": "application/json",
        "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
      }
    });
    console.log(response.data); // Returns job_id immediately
  } catch (error) {
    console.error("Error triggering workflow:", error);
  }
};

runWorkflow();
curl -X POST \
"$ASTRA_MULTIAGENT_AI_SERVER_URL/api/v2/workflows" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \
-d '{
  "flow_id": "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  "background": true,
  "stream": false,
  "inputs": {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "Process this in the background",
    "ChatInput-abc.session_id": "session-456"
  }
}'
```

Ответ:

```
{
  "job_id": "job_id_1234567890",
  "created_timestamp": "2025-01-15T10:30:00Z",
  "status": "queued",
  "errors": []
}
```

Тело запроса

Поле	Тип	Обязатель- но	По умолчанию	Описание
flow_id	string	Yes	-	Идентификатор потока или имя его конечной точки.
flow_version	string	No	-	Необязательный хеш версии для привязки к конкретной версии потока.
background	boolean	No	false	Если false, рабочий

Поле	Тип	Обязатель- но	По умолчанию	Описание
inputs	object	No	{}	процесс выполняется синхронно. Входные данные для выполнения рабочего процесса. Используются идентификатор ы компонентов с точечной нотацией, например ChatInput- abc.input_value. См. Идентификато ры компонентов и структура ввода.

Пример ответа

```
{
  "flow_id": "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  "job_id": "job_id_1234567890",
  "object": "response",
  "created_at": 1741476542,
  "status": "completed",
  "errors": [],
  "inputs": {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "what is 2+2",
    "ChatInput-abc.session_id": "session-123"
  },
  "outputs": {
    "ChatOutput-xyz": {
      "type": "message",
      "component_id": "ChatOutput-xyz",
      "status": "completed",
      "content": "2 + 2 equals 4."
    }
  },
  "metadata": {}
}
```

Тело ответа

Ответ включает поле outputs, содержащее результаты на уровне компонентов. Каждый вывод содержит поле type, указывающее тип содержимого.

Тип	Описание	Пример
message	Текстовое содержимое сообщения.	Ответы чата, резюме
image	URL изображения или сами данные изображения.	Сгенерированные или обработанные изображения
sql	Результаты SQL-запроса.	Выводы запросов к базе данных
data	Структурированные данные.	JSON-объекты, массивы
file	Ссылка на файл.	Сгенерированные документы, отчёты

Конечная точка получения статуса рабочего процесса

Конечная точка: GET /api/v2/workflows

Описание: возвращает статус и результаты выполнения рабочего процесса по идентификатору задания.

Пример запроса

```
url = f"{ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows"
params = {
    "job_id": "job_id_1234567890"
}
headers = {
    "accept": "application/json",
    "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
}
```

```
response = requests.get(url, params=params, headers=headers)
print(response.json())
```

```
const jobId = "job_id_1234567890";
const url = `${ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows`;
```

```
const getWorkflowStatus = async () => {
  try {
    const response = await axios.get(url, {
      params: {
        job_id: jobId
      },
      headers: {
        "accept": "application/json",
        "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
      }
    });
    console.log(response.data);
  } catch (error) {
    console.error("Error getting workflow status:", error);
  }
};
```

```
getWorkflowStatus();
```

```
curl -X GET \
  "$ASTRA_MULTIAGENT_AI_SERVER_URL/api/v2/workflows?job_id=job_id_1234567890" \
  -H "accept: application/json" \
  -H "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY"
```

Параметры запроса

Параметр	Тип	Обязательно	Описание
job_id	string	Yes	Идентификатор задания, возвращённый после запуска рабочего процесса.
stream	boolean	No	Если true, возвращает поток серверных событий. По умолчанию — false.
sequence_id	integer	No	Необязательный идентификатор последовательности для возобновления потока с определённого места.

Пример ответа

```
{
  "flow_id": "flow_67ccd2be17f0819081ff3bb2cf6508e60bb6a6b452d3795b",
  "job_id": "job_id_1234567890",
  "object": "response",
  "created_at": 1741476542,
  "status": "completed",
  "errors": [],
  "outputs": {
    "ChatOutput-xyz": {
      "type": "message",
      "component_id": "ChatOutput-xyz",
      "status": "completed",
      "content": "Processing complete..."
    }
  },
  "input": [
    {
      "type": "text",
      "data": "Input text prompt for the workflow execution",
      "role": "User"
    }
  ],
  "metadata": {}
}
```

Тело ответа

Ответ включает поле status, которое показывает текущее состояние выполнения рабочего процесса.

Статус	Описание
queued	Задание поставлено в очередь и ожидает запуска.
in_progress	Задание выполняется.
completed	Задание успешно завершено.
failed	Задание завершилось с ошибкой.
error	Во время выполнения возникла ошибка.

Конечная точка остановки рабочего процесса

Конечная точка: POST /api/v2/workflows/stop

Описание: останавливает запущенное выполнение рабочего процесса по идентификатору задания.

Пример запроса

```
url = f"{ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows/stop"
headers = {
    "Content-Type": "application/json",
    "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
}
payload = {
    "job_id": "job_id_1234567890"
}

response = requests.post(url, json=payload, headers=headers)
print(response.json())

const url = `${ASTRA_MULTIAGENT_AI_SERVER_URL}/api/v2/workflows/stop`;

const payload = {
  job_id: "job_id_1234567890"
};

const stopWorkflow = async () => {
  try {
    const response = await axios.post(url, payload, {
      headers: {
        "Content-Type": "application/json",
        "x-api-key": ASTRA_MULTIAGENT_AI_API_KEY
      }
    });
    console.log(response.data);
  } catch (error) {
    console.error("Error stopping workflow:", error);
  }
};

stopWorkflow();
curl -X POST \
"$ASTRA_MULTIAGENT_AI_SERVER_URL/api/v2/workflows/stop" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_MULTIAGENT_AI_API_KEY" \
-d '{
  "job_id": "job_id_1234567890"
}'
```

Тело запроса

Поле	Тип	Обязатель но	По умолчанию	Описание
job_id	string	Yes	-	Идентификатор задания рабочего процесса, которое необходимо остановить.

Пример ответа

```
{
  "job_id": "job_id_1234567890",
  "message": "Job job_id_1234567890 cancelled successfully."
}
```

Идентификаторы компонентов и структура ввода

Workflow API использует идентификаторы компонентов с точечной нотацией для указания входных данных отдельных компонентов рабочего процесса. Это позволяет передавать значения в конкретные компоненты и переопределять параметры компонентов.

Идентификаторы компонентов используют формат {component_id}.{parameter_name}. При выполнении запросов Workflow API включайте идентификаторы компонентов в объект inputs. Например, следующий запрос показывает, как адресовать несколько компонентов и их параметров одновременно.

```
{
  "flow_id": "your-flow-id",
  "inputs": {
    "ChatInput-abc.input_type": "chat",
    "ChatInput-abc.input_value": "what is 2+2",
    "ChatInput-abc.session_id": "session-123",
    "OpenSearchComponent-xyz.opensearch_url": "https://opensearch:9200",
    "LLMComponent-123.temperature": 0.7,
    "LLMComponent-123.max_tokens": 100
  }
}
```

Чтобы найти идентификатор компонента в интерфейсе Генеративный ИИ, откройте поток, выберите компонент и нажмите **Controls**. Идентификатор компонента отображается в верхней части панели **Controls**.

Вы можете переопределять параметры любого компонента.

Обработка ошибок

API использует стандартные HTTP-коды состояния для обозначения успешного выполнения запроса или ошибки.

Код состояния	Описание
200 OK	Запрос выполнен успешно.
400 Bad Request	Переданы некорректные параметры запроса.
401 Unauthorized	API-ключ отсутствует или недействителен.
404 Not Found	Поток не найден или Developer API отключён.
500 Internal Server Error	Во время выполнения произошла ошибка сервера.
501 Not Implemented	Конечная точка ещё не реализована.

Формат ответа с ошибкой

```
{
  "detail": "Error message describing what went wrong"
}
```

31. OpenAI Responses API

Генеративный ИИ включает конечную точку, совместимую с OpenAI Responses API. Она доступна по адресу POST /api/v1/responses.

Эта конечная точка позволяет использовать существующие клиентские библиотеки OpenAI с минимальными изменениями кода. Вам нужно лишь заменить имя model, например gpt-4, на ваш FLOW_ID. Идентификаторы потоков (Flow IDs) можно найти в примерах кода на панели **API доступа** или в URL потока.

31.1 Предварительные требования

Для совместимости с конечной точкой OpenAI Responses API в модуле «Генеративный ИИ» ваш поток и запрос должны соответствовать следующим требованиям:

- **Ввод в чат:** Поток должен содержать компонент **Ввод в чат**. Потоки без этого компонента возвращают ошибку при вызове этой конечной точки. Типы компонентов ChatInput и Chat Input распознаются как ввод чата.
- **Инструменты:** Параметр tools не поддерживается и приведёт к ошибке, если указан.
- **Название модели:** В поле model запроса должен быть указан действительный идентификатор потока или имя конечной точки.
- **Аутентификация:** Все запросы требуют API-ключа, передаваемого в заголовке x-api-key. Для получения дополнительной информации см. Ключи API и аутентификация.

Дополнительная конфигурация для клиентских библиотек OpenAI

Эта конечная точка совместима с API OpenAI, но требует специальной конфигурации при использовании клиентских библиотек OpenAI. Генеративный ИИ использует заголовок x-api-key для аутентификации, тогда как OpenAI использует заголовок Authorization: Bearer. При отправке запросов в модуле «Генеративный ИИ» через клиентские библиотеки OpenAI необходимо задать пользовательские заголовки и включить параметр api_key. Значение api_key может быть произвольным, например "dummy-api-key", поскольку фактическая аутентификация выполняется через default_headers.

В следующих примерах замените значения ASTRA_SERVER_URL, ASTRA_API_KEY и FLOW_ID на значения вашего развёртывания.

```
from openai import OpenAI
```

```
client = OpenAI(
    base_url="ASTRA_SERVER_URL/api/v1/",
    default_headers={"x-api-key": "ASTRA_API_KEY"},
    api_key="dummy-api-key" # Требуется SDK OpenAI, но не используется Генеративный ИИ
)
```

```
response = client.responses.create(
    model="FLOW_ID",
    input="There is an event that happens on the second wednesday of every month. What are the event dates in 2026?",
)
```

```
print(response.output_text)
```

```
const client = new OpenAI({
  baseURL: "ASTRA_SERVER_URL/api/v1/",
  defaultHeaders: {
    "x-api-key": "ASTRA_API_KEY",
  },
});
```

```
apiKey: "dummy-api-key", // Требуется SDK OpenAI, но не используется Генеративный ИИ
});

const response = await client.responses.create({
  model: "FLOW_ID",
  input:
    "There is an event that happens on the second wednesday of every month. What are the event dates in 2026?",
});

console.log(response.output_text);
```

Ответ

Here are the event dates for the second Wednesday of each month in 2026:

- January 14, 2026
- February 11, 2026
- March 11, 2026
- April 8, 2026
- May 13, 2026
- June 10, 2026
- July 8, 2026
- August 12, 2026
- September 9, 2026
- October 14, 2026
- November 11, 2026
- December 9, 2026

31.2 Пример запроса

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/responses" \
-H "x-api-key: $ASTRA_API_KEY" \
-H "Content-Type: application/json" \
-d '{
  "model": "YOUR_FLOW_ID",
  "input": "Hello, how are you?",
  "stream": false
}'
```

Заголовки

Заголовок	Обязательно	Описание	Пример
x-api-key	Да	Ваш API-ключ Генеративный ИИ для аутентификации	"sk-..."
Content-Type	Да	Формат тела запроса JSON	"application/json"
X-ASTRA- GLOBAL-VAR-*	Нет	Глобальные переменные для потока	X-ASTRA- GLOBAL-VAR- API_KEY: sk-... (подробности см. Передача глобальных переменных)

Тело запроса

Поле	Тип	Обязатель- но	По умолчанию	Описание
model	string	Да	—	Идентификатор потока или имя конечной точки.
input	string	Да	—	Входной текст для обработки.
stream	boolean	Нет	false	Включить потоковую передачу ответа.
background	boolean	Нет	false	Выполнять обработку в фоновом режиме.
tools	list[Any]	Нет	null	Инструменты не поддерживаются.
previous_response_id	string	Нет	null	Идентификатор предыдущего ответа для продолжения беседы (см. Продолжение бесед).
include	list[string]	Нет	null	Дополнительные данные ответа, например ["tool_call.results"] (см. Результаты вызова инструмента).

31.3 Пример ответа

```
{
  "id": "e5e8ef8a-7efd-4090-a110-6aca082bceb7",
  "object": "response",
  "created_at": 1756837941,
  "status": "completed",
```

```

"model": "ced2ec91-f325-4bf0-8754-f3198c2b1563",
"output": [
  {
    "type": "message",
    "id": "msg_e5e8ef8a-7efd-4090-a110-6aca082bceb7",
    "status": "completed",
    "role": "assistant",
    "content": [
      {
        "type": "output_text",
        "text": "Hello! I'm here and ready to help. How can I assist you today?",
        "annotations": []
      }
    ]
  }
],
"parallel_tool_calls": true,
"previous_response_id": null,
"reasoning": { "effort": null, "summary": null },
"store": true,
"temperature": 1.0,
"text": { "format": { "type": "text" } },
"tool_choice": "auto",
"tools": [],
"top_p": 1.0,
"truncation": "disabled",
"usage": null,
"user": null,
"metadata": {}
}

```

Поля ответа

Динамические поля Генеративный ИИ:

Поле	Тип	Описание
id	string	Уникальный идентификатор ответа.
created_at	int	Unix-временная метка создания ответа.
model	string	Идентификатор потока, который был выполнен.
output	list[dict]	Массив элементов вывода (сообщения, вызовы инструментов и т.д.).
previous_response_id	string null	Идентификатор предыдущего ответа, если продолжается диалог.
usage	dict null	Статистика использования токенов (prompt_tokens, completion_tokens, total_tokens), если доступна.

Поля со значениями по умолчанию, совместимыми с OpenAI:

Поле	Тип	Значение по умолчанию	Описание
object	string	"response"	Тип объекта.

Поле	Тип	Значение по умолчанию	Описание
status	string	"completed"	Статус ответа ("completed", "in_progress", "failed").
error	dict null	null	Детали ошибки (если есть).
incomplete_details	dict null	null	Детали неполного ответа (если есть).
instructions	string null	null	Инструкции (если есть).
max_output_tokens	int null	null	Ограничение на токены вывода (если есть).
parallel_tool_calls	boolean	true	Включены ли параллельные вызовы инструментов.
reasoning	dict	{"effort": null, "summary": null}	Метаданные рассуждений.
store	boolean	true	Сохраняется ли ответ.
temperature	float	1.0	Температура выборки.
text	dict	{"format": {"type": "text"}}	Конфигурация текстового вывода.
tool_choice	string	"auto"	Режим выбора инструмента.
tools	list[dict]	[]	Описания доступных инструментов.
top_p	float	1.0	Параметр top-p.
truncation	string	"disabled"	Стратегия усечения.
user	string null	null	Идентификатор пользователя (если есть).
metadata	dict	{}	Дополнительные метаданные.

31.4 Пример запроса с потоковой передачей

```
curl -X POST \
  "$ASTRA_SERVER_URL/api/v1/responses" \
  -H "x-api-key: $ASTRA_API_KEY" \
```

```
-H "Content-Type: application/json" \
-d '{
  "model": "FLOW_ID",
  "input": "Tell me a story about a robot",
  "stream": true
}'
```

Результат (фрагмент)

```
{
  "id": "f7fcea36-f128-41c4-9ac1-e683137375d5",
  "object": "response.chunk",
  "created": 1756838094,
  "model": "ced2ec91-f325-4bf0-8754-f3198c2b1563",
  "delta": {
    "content": "Once"
  },
  "status": null
}
```

Тело потокового ответа

Поле	Тип	Описание
id	string	Уникальный идентификатор ответа.
object	string	Всегда "response.chunk".
created	int	Unix-временная метка создания фрагмента.
model	string	Идентификатор потока.
delta	dict	Новый фрагмент содержимого.
status	string null	Статус ("completed", "in_progress", "failed") или null.

Поток продолжается до финального фрагмента со статусом "completed".

Финальный фрагмент

```
{
  "id": "f7fcea36-f128-41c4-9ac1-e683137375d5",
  "object": "response.chunk",
  "created": 1756838094,
  "model": "ced2ec91-f325-4bf0-8754-f3198c2b1563",
  "delta": {},
  "status": "completed"
}
```

31.5 Продолжение бесед с previous_response_id

Непрерывность диалога позволяет сохранять контекст между несколькими вызовами API.

Первое сообщение:

```
curl -X POST \
  "$ASTRA_SERVER_URL/api/v1/responses" \
  -H "x-api-key: $ASTRA_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "model": "FLOW_ID",
    "input": "Hello, my name is Alice"
  }'
```

Результат


```
{
  "id": "c45f4ac8-772b-4675-8551-c560b1afd590",
  "object": "response",
  "created_at": 1756839042,
  "status": "completed",
  "model": "ced2ec91-f325-4bf0-8754-f3198c2b1563",
  "output": [
    {
      "type": "message",
      "id": "msg_c45f4ac8-772b-4675-8551-c560b1afd590",
      "status": "completed",
      "role": "assistant",
      "content": [
        {
          "type": "output_text",
          "text": "Hello, Alice! How can I assist you today?",
          "annotations": []
        }
      ]
    }
  ],
  "previous_response_id": null
}
```

Последующее сообщение:

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/responses" \
-H "x-api-key: $ASTRA_API_KEY" \
-H "Content-Type: application/json" \
-d '{
  "model": "ced2ec91-f325-4bf0-8754-f3198c2b1563",
  "input": "What's my name?",
  "previous_response_id": "c45f4ac8-772b-4675-8551-c560b1afd590"
}'
```

Вы также можете использовать собственные идентификаторы сессий как `previous_response_id`:

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/responses" \
-H "x-api-key: $ASTRA_API_KEY" \
-H "Content-Type: application/json" \
-d '{
  "model": "ced2ec91-f325-4bf0-8754-f3198c2b1563",
  "input": "What's my name?",
  "previous_response_id": "session-alice-1756839048"
}'
```

(Пример ответа остаётся таким, как в исходном тексте.)

31.6 Получение результатов вызова инструмента

Чтобы вернуть необработанные результаты выполнения инструментов, добавьте `"include": ["tool_call.results"]` в тело запроса:

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/responses" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "model": "FLOW_ID",
  "input": "Calculate 23 * 15 and show me the result",
  "stream": false,
  "include": ["tool_call.results"]
}'
```

(Примеры структур `tool_call` и полного ответа оставлены без изменений по сути, только выровнены формально.)

31.7 Передача глобальных переменных через заголовки

Глобальные переменные позволяют передавать динамические значения (API-ключи, идентификаторы пользователей и т.п.) в поток.

Конечная точка `/responses` принимает глобальные переменные через заголовки формата `X-ASTRA-GLOBAL-VAR-{VARIABLE_NAME}`. Имена переменных автоматически приводятся к верхнему регистру, значения действуют только в рамках текущего запроса.

Пример:

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/responses" \
-H "x-api-key: $ASTRA_API_KEY" \
-H "Content-Type: application/json" \
-H "X-ASTRA-GLOBAL-VAR-OPENAI_API_KEY: sk-..." \
-H "X-ASTRA-GLOBAL-VAR-USER_ID: user123" \
-H "X-ASTRA-GLOBAL-VAR-ENVIRONMENT: production" \
-d '{
  "model": "your-flow-id",
  "input": "Hello"
}'
```

Поведение при отсутствии переменных можно дополнительно настроить с помощью `FALLBACK_TO_ENV_VARS=true`.

31.8 Отслеживание использования токенов

Поле `usage` заполняется, если компоненты потока возвращают информацию о токенах. Оно совместимо с форматом OpenAI:

```
"usage": {
  "prompt_tokens": 12,
  "completion_tokens": 145,
  "total_tokens": 157
}
```

Поле всегда присутствует в ответе: либо с подсчётами, либо `null`.

(Примеры на Python, TypeScript и curl из исходного текста остаются — в них заменены только плейсхолдеры `ASTRA_SERVER_URL_URL/...` на `ASTRA_SERVER_URL` и `ASTRA_API_KEY` на `ASTRA_API_KEY`.)

32. Управление потоком

Используйте эндпоинт /flows для создания, чтения, обновления и удаления потоков.

Если вы хотите использовать API Генеративный ИИ для запуска потока, см. Эндпоинты запуска потоков.

32.1 Создать поток

Создаёт новый поток.

```
curl -X POST \
"$ASTRA_URL/api/v1/flows/" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "name": "string2",
  "description": "string",
  "icon": "string",
  "icon_bg_color": "#FF0000",
  "gradient": "string",
  "data": {},
  "is_component": false,
  "updated_at": "2024-12-30T15:48:01.519Z",
  "webhook": false,
  "endpoint_name": "string",
  "tags": [
    "string"
  ]
}'
```

Результат

```
{
  "name": "string2",
  "description": "string",
  "icon": "string",
  "icon_bg_color": "#FF0000",
  "gradient": "string",
  "data": {},
  "is_component": false,
  "updated_at": "2025-02-04T21:07:36+00:00",
  "webhook": false,
  "endpoint_name": "string",
  "tags": ["string"],
  "locked": false,
  "id": "e8d81c37-714b-49ae-ba82-e61141f020ee",
  "user_id": "f58396d4-a387-4bb8-b749-f40825c3d9f3",
  "project_id": "1415de42-8f01-4f36-bf34-539f23e47466"
}
```

32.2 Создать потоки

Создаёт несколько новых потоков, возвращая массив объектов потоков.

```
curl -X POST \
"$ASTRA_URL/api/v1/flows/batch/" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "flows": [
```

```
{
  "name": "string",
  "description": "string",
  "icon": "string",
  "icon_bg_color": "string",
  "gradient": "string",
  "data": {},
  "is_component": false,
  "updated_at": "2024-12-30T18:36:02.737Z",
  "webhook": false,
  "endpoint_name": "string",
  "tags": [
    "string"
  ],
  "locked": false,
  "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "project_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
},
{
  "name": "string",
  "description": "string",
  "icon": "string",
  "icon_bg_color": "string",
  "gradient": "string",
  "data": {},
  "is_component": false,
  "updated_at": "2024-12-30T18:36:02.737Z",
  "webhook": false,
  "endpoint_name": "string",
  "tags": [
    "string"
  ],
  "locked": false,
  "user_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "project_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
}
]
```

32.3 Прочитать поток

Получает конкретный поток по его ID.

```
curl -X GET \
"$ASTRA_URL/api/v1/flows/$FLOW_ID" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "name": "Basic Prompting",
  "description": "Perform basic prompting with an OpenAI model.",
  "icon": "Braces",
  "icon_bg_color": null,
  "gradient": "2",
  "data": {
    "nodes": [
      ...
    ]
  }
}
```

32.4 Прочитать потоки

Возвращает объект JSON, содержащий список потоков.

Получите все потоки с пагинацией:

```
curl -X GET \
```

```
"$ASTRA_URL/api/v1/flows/?remove_example_flows=false&components_only=false&get_all=true&header_flows=false&page=1&size=50" \
```

```
-H "accept: application/json" \
```

```
-H "x-api-key: $ASTRA_API_KEY"
```

Чтобы получить потоки из конкретного проекта, используйте параметр запроса `project_id`:

```
curl -X GET \
```

```
"$ASTRA_URL/api/v1/flows/?remove_example_flows=true&components_only=false&get_all=false&project_id=$PROJECT_ID&header_flows=false&page=1&size=1" \
```

```
-H "accept: application/json" \
```

```
-H "x-api-key: $ASTRA_API_KEY"
```

32.5 Прочитать образцы потоков

Получает список образцов потоков:

```
curl -X GET \
```

```
"$ASTRA_URL/api/v1/flows/basic_examples/" \
```

```
-H "accept: application/json" \
```

```
-H "x-api-key: $ASTRA_API_KEY"
```

32.6 Обновить поток

Обновляет существующий поток по его ID.

В этом примере значение `endpoint_name` изменяется с случайного UUID на `my_new_endpoint_name`.

```
curl -X PATCH \
```

```
"$ASTRA_URL/api/v1/flows/$FLOW_ID" \
```

```
-H "accept: application/json" \
```

```
-H "Content-Type: application/json" \
```

```
-H "x-api-key: $ASTRA_API_KEY" \
```

```
-d '{
```

```
  "name": "string",
```

```
  "description": "string",
```

```
  "data": {},
```

```
  "project_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
```

```
  "endpoint_name": "my_new_endpoint_name",
```

```
  "locked": true
```

```
}'
```

Результат

```
{
```

```
  "name": "string",
```

```
  "description": "string",
```

```
  "icon": "Braces",
```

```
  "icon_bg_color": null,
```

```
  "gradient": "2",
```

```
  "data": {},
```

```
  "is_component": false,
```

```
  "updated_at": "2024-12-30T18:30:22+00:00",
```

```
  "webhook": false,
```

```
  "endpoint_name": "my_new_endpoint_name",
```

```
  "tags": null,
```

```
  "locked": true,
```

```
{
  "id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
  "user_id": "f58396d4-a387-4bb8-b749-f40825c3d9f3",
  "project_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"
}
```

32.7 Удалить поток

Удаляет конкретный поток по его ID.

```
curl -X DELETE \
"$ASTRA_URL/api/v1/flows/$FLOW_ID" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "message": "Flow deleted successfully"
}
```

32.8 Экспортировать потоки

Экспортирует указанные потоки в ZIP-файл.

Этот эндпоинт загружает ZIP-файл, содержащий JSON-файлы Генеративный ИИ для каждого ID потока, указанного в теле запроса.

```
curl -X POST \
"$ASTRA_URL/api/v1/flows/download/" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '[
  "e1e40c77-0541-41a9-88ab-ddb3419398b5",
  "92f9a4c5-cfc8-4656-ae63-1f0881163c28"
]' \
--output astra-flows.zip
```

Результат

% Total	% Received	% Xferd	Average Speed	Time	Time	Time	Current
			Dload Upload	Total	Spent	Left	Speed
100	76437	0	76353	100	84	4516k	5088 ---:--:-- ---:--:-- ---:--:-- 4665k

32.9 Импортировать потоки

Импортирует потоки, загружая JSON-файл, совместимый с модулем «Генеративный ИИ».

Чтобы указать целевой проект для потока, включите параметр запроса `folder_id`. Целевой `folder_id` должен уже существовать до загрузки потока. Вызовите эндпоинт `/api/v1/projects/` для получения списка доступных папок и проектов.

В этом примере загружается локальный файл с именем `agent-with-astra-db-tool.json` в папку, указанную переменной `FOLDER_ID`:

```
curl -X POST \
"$ASTRA_URL/api/v1/flows/upload/?folder_id=$FOLDER_ID" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-H "x-api-key: $ASTRA_API_KEY" \
-F "file=@agent-with-astra-db-tool.json;type=application/json"
```

Результат

```
[
  {
    "name": "agent-with-astra-db-tool",
    "description": ""
  }
]
```

```
"icon": null,  
"icon_bg_color": null,  
"gradient": null,  
"data": {}  
...  
}  
]
```

33. Файлы

Используйте конечные точки /files, чтобы перемещать файлы между вашим локальным компьютером и Генеративный ИИ.

Все конечные точки /files (как /v1/files, так и /v2/files) требуют аутентификации с помощью ключа API Генеративный ИИ. Вы можете получать доступ только к файлам, принадлежащим вашей учетной записи, даже будучи суперпользователем.

33.1 Различия между /v1/files и /v2/files

Существует две версии конечных точек /files.

/v2/files предлагает следующие улучшения по сравнению с /v1/files:

- Файлы /v2 организованы по user_id вместо flow_id. Это означает, что файлы принадлежат пользователям и не привязаны к конкретным потокам. Вы можете загрузить файл в модуле «Генеративный ИИ» один раз и использовать его в нескольких потоках.
- Файлы /v2 отслеживаются в базе данных Генеративный ИИ.
- /v2 поддерживает массовую загрузку и удаление.
- Ответы /v2 содержат более подробные метаданные.

Однако /v2/files не поддерживает файлы изображений. Чтобы отправлять файлы изображений в ваши потоки через API, используйте загрузку файлов изображений (v1).

33.2 Конечные точки файлов (v1)

Используйте конечные точки /files для перемещения файлов между вашим локальным компьютером и Генеративный ИИ.

Загрузка файла (v1)

Загрузите файл в конечную точку v1/files/upload/\$FLOW_ID: Замените **FILE_NAME** именем загружаемого файла.

```
curl -X POST \
"$ASTRA_URL/api/v1/files/upload/$FLOW_ID" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-H "x-api-key: $ASTRA_API_KEY" \
-F "file=@FILE_NAME.txt"
```

Замените FILE_NAME.txt именем и расширением файла, который хотите загрузить. Не все типы файлов поддерживаются.

Результат

```
{
  "flowId": "92f9a4c5-cfc8-4656-ae63-1f0881163c28",
  "file_path": "92f9a4c5-cfc8-4656-ae63-1f0881163c28/2024-12-30_15-19-43_your_file.txt"
}
```

Загрузка файлов изображений (v1)

Отправьте файлы изображений в модуле «Генеративный ИИ» для использования их в потоках.

Ограничение по умолчанию для файлов составляет 1024 МБ. Чтобы изменить это ограничение, установите ASTRA_MAX_FILE_SIZE_UPLOAD переменную окружения.

1. Приложите изображение к запросу POST /v1/files/upload/\$FLOW_ID с --form (-F) и указанием пути к файлу:

```
curl -X POST "$ASTRA_URL/api/v1/files/upload/$FLOW_ID" \
-H "Content-Type: multipart/form-data" \
-H "x-api-key: $ASTRA_API_KEY" \
-F "file=@PATH/TO/FILE.png"
```

Успешный запрос возвращает file_path изображения в системе управления файлами Генеративный ИИ в формате FLOW_ID/TIMESTAMP_FILENAME.TYPE. Например:

```
{
  "flowId": "a430cc57-06bb-4c11-be39-d3d4de68d2c4",
  "file_path": "a430cc57-06bb-4c11-be39-d3d4de68d2c4/2024-11-27_14-47-50_image-file.png"
}
```

2. Используйте возвращённый file_path для отправки файла изображения другим компонентам, которые могут принимать ввод файлов. Где указывается путь к файлу, зависит от типа компонента.

В следующем примере запускается шаблонный поток **Basic Prompting**, передавая файл изображения и запрос describe this image в качестве входных данных для компонента

Ввод в чат. В данном случае путь к файлу указывается в tweaks.

```
curl -X POST \
"$ASTRA_URL/api/v1/run/a430cc57-06bb-4c11-be39-d3d4de68d2c4?stream=false" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "output_type": "chat",
  "input_type": "chat",
  "tweaks": {
    "ChatInput-b67sL": {
      "files": "a430cc57-06bb-4c11-be39-d3d4de68d2c4/2024-11-27_14-47-50_image-file.png",
      "input_value": "describe this image"
    }
  }
}'
```

Для получения помощи с настройками используйте ****Input Schema**** в панели ****доступ к API**** потока.

Настройка параметров с помощью **Input Schema** также автоматически заполняет необходимые идентификаторы компонентов. ### Список файлов (v1)

Выведите список всех файлов, связанных с конкретным потоком.

```
curl -X GET \
"$ASTRA_URL/api/v1/files/list/$FLOW_ID" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "files": ["2024-12-30_15-19-43_your_file.txt"]
}
```

Загрузка файла (v1)

Скачайте конкретный файл из потока.

```
curl -X GET \
"$ASTRA_URL/api/v1/files/download/$FLOW_ID/2024-12-30_15-19-43_your_file.txt" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
--output downloaded_file.txt
```

Результат

File contents downloaded to downloaded_file.txt

Удаление файла (v1)

Удалите конкретный файл из потока.

```
curl -X DELETE \
"$ASTRA_URL/api/v1/files/delete/$FLOW_ID/2024-12-30_15-19-43_your_file.txt" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
Результат
{
  "message": "File 2024-12-30_15-19-43_your_file.txt deleted successfully"
}
```

33.3 Конечные точки файлов (v2)

Используйте конечные точки /files для перемещения файлов между вашей локальной машиной и Генеративный ИИ.

Конечные точки /v2/files могут быть аутентифицированы с помощью API-ключа или JWT. Чтобы создать API-ключ Генеративный ИИ и экспортировать его как переменную окружения, см. Начало работы с модулем «Генеративный ИИ» API.

Загрузка файла (v2)

Загрузите файл в свою учетную запись пользователя. Файл может использоваться в нескольких потоках.

Файл загружается в формате USER_ID/FILE_ID.FILE_EXTENSION, например 07e5b864-e367-4f52-b647-a48035ae7e5e/d44dc2e1-9ae9-4cf6-9114-8d34a6126c94.pdf.

1. Чтобы получить ваш текущий user_id, вызовите конечную точку /whoami:

```
curl -X GET \
"$ASTRA_URL/api/v1/users/whoami" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
Результат
{"id":"07e5b864-e367-4f52-b647-a48035ae7e5e","username":"astra","profile_image":null,"store_api_key":null,"is_active":true,"is_superuser":true,"create_at":"2025-05-08T17:59:07.855965","updated_at":"2025-05-28T19:00:42.556460","last_login_at":"2025-05-28T19:00:42.554338","optins":{"github_starred":false,"dialog_dismissed":true,"discord_clicked":false,"mcp_dialog_dismissed":true}}
```

2. В POST-запросе к v2/files замените @FILE_NAME.EXTENSION на имя загружаемого файла и его расширение. Вы должны включить амперсанд (@) в запрос, чтобы curl загрузил содержимое файла, а не строку FILE_NAME.EXTENSION.

```
curl -X POST \
"$ASTRA_URL/api/v2/files" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-H "x-api-key: $ASTRA_API_KEY" \
-F "file=@FILE_NAME.EXTENSION"
```

Файл загружается в формате USER_ID/FILE_ID.FILE_EXTENSION, и API возвращает метаданные о загруженном файле:

```
{
  "id":"d44dc2e1-9ae9-4cf6-9114-8d34a6126c94",
  "name":"engine_manual",
  "path":"07e5b864-e367-4f52-b647-a48035ae7e5e/d44dc2e1-9ae9-4cf6-9114-8d34a6126c94.pdf",
  "size":851160,
  "provider":null
}
```

Отправка файлов в ваши потоки (v2)

Конечная точка /v2/files не может отправлять изображения в потоки. Чтобы отправлять изображения в потоки через API, см. Upload image files (v1). Эта конечная точка загружает файлы в систему управления файлами вашего сервера Генеративный ИИ. Чтобы использовать

загруженный файл в потоке, передайте путь к файлу в поток с помощью компонента **Чтение файла**.

Ограничение по умолчанию составляет 1024 МБ. Чтобы изменить это значение, измените переменную окружения `ASTRA_MAX_FILE_SIZE_UPLOAD` environment variable.

1. Чтобы отправить файл в ваш поток через API, выполните POST-запрос к конечной точке `/api/v2/files`.

Замените **FILE_NAME.EXTENSION** на имя и расширение файла, который хотите загрузить. Это тот же шаг, описанный в Upload file (v2), но поскольку вам нужен файл для загрузки в поток, он включён здесь.

```
curl -X POST \
"$ASTRA_URL/api/v2/files" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-H "x-api-key: $ASTRA_API_KEY" \
-F "file=@FILE_NAME.EXTENSION"
```

Файл загружается в формате `USER_ID/FILE_ID.FILE_EXTENSION`, и API возвращает метаданные о загруженном файле:

```
{
  "id": "d44dc2e1-9ae9-4cf6-9114-8d34a6126c94",
  "name": "engine_manual",
  "path": "07e5b864-e367-4f52-b647-a48035ae7e5e/d44dc2e1-9ae9-4cf6-9114-8d34a6126c94.pdf",
  "size": 851160,
  "provider": null
}
```

2. Чтобы использовать этот файл в вашем потоке, добавьте компонент **Чтение файла** в ваш поток. Этот компонент загружает файлы в потоки с вашей локальной машины или из системы управления файлами Генеративный ИИ.

3. Запустите поток, передав path в компонент Read-File объекта tweaks:

```
curl --request POST \
--url "$ASTRA_URL/api/v1/run/$FLOW_ID" \
--header "Content-Type: application/json" \
--header "x-api-key: $ASTRA_API_KEY" \
--data '{
  "input_value": "what do you see?",
  "output_type": "chat",
  "input_type": "text",
  "tweaks": {
    "Read-File-1olS3": {
      "path": [
        "07e5b864-e367-4f52-b647-a48035ae7e5e/3a290013-fe1e-4d3d-a454-cacae81288f3.pdf"
      ]
    }
  }
}'
```

Чтобы получить ID компонента Read-File, вызовите конечную точку читать поток или проверьте компонент в визуальном редакторе.

Если путь к файлу действителен, поток выполнится успешно.

Список файлов (v2)

Выведите список всех файлов, связанных с вашей учетной записью пользователя.

```
curl -X GET \
"$ASTRA_URL/api/v2/files" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
[
{
  "id": "c7b22c4c-d5e0-4ec9-af97-5d85b7657a34",
  "name": "your_file",
  "path": "6f17a73e-97d7-4519-a8d9-8e4c0be411bb/c7b22c4c-d5e0-4ec9-af97-5d85b7657a34.txt",
  "size": 1234,
  "provider": null
}
]
```

Скачивание файла (v2)

Скачайте конкретный файл по его ID и расширению.

Необходимо указать ожидаемый тип файла в значении --output.

```
curl -X GET \
"$ASTRA_URL/api/v2/files/c7b22c4c-d5e0-4ec9-af97-5d85b7657a34" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
--output downloaded_file.txt
```

Результат

File contents downloaded to downloaded_file.txt

Изменение имени файла (v2)

Измените имя файла.

```
curl -X PUT \
"$ASTRA_URL/api/v2/files/$FILE_ID?name=new_file_name" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "id": "76543e40-f388-4cb3-b0ee-a1e870aca3d3",
  "name": "new_file_name",
  "path": "6f17a73e-97d7-4519-a8d9-8e4c0be411bb/76543e40-f388-4cb3-b0ee-a1e870aca3d3.png",
  "size": 2728251,
  "provider": null
}
```

Удаление файла (v2)

Удалите конкретный файл по его ID.

```
curl -X DELETE \
"$ASTRA_URL/api/v2/files/$FILE_ID" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "message": "File deleted successfully"
}
```

Удаление всех файлов (v2)

Удалите все файлы, связанные с вашей учетной записью пользователя.

```
curl -X DELETE \
"$ASTRA_URL/api/v2/files" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "message": "All files deleted successfully"
}
```

33.4 Создание загрузки файла (Устарело)

Эта конечная точка устарела. Вместо неё используйте конечные точки /files.

34. Проекты

Используйте конечную точку `/projects` для создания, чтения, обновления и удаления проектов Генеративный ИИ.

34.1 Чтение проектов

Получите список проектов Генеративный ИИ, включая их идентификаторы, названия и описания.

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/projects/" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"

Результат

[
{
  "name": "Starter Project",
  "description": "Manage your own projects. Download and upload projects.",
  "id": "1415de42-8f01-4f36-bf34-539f23e47466",
  "parent_id": null
}
]
```

34.2 Создание проекта

Создайте новый проект.

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/projects/" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "name": "new_project_name",
  "description": "string",
  "components_list": [],
  "flows_list": []
}'

Результат

{
  "name": "new_project_name",
  "description": "string",
  "id": "b408ddb9-6266-4431-9be8-e04a62758331",
  "parent_id": null
}
```

Чтобы добавить потоки и компоненты при создании проекта, получите значения `components_list` и `flows_list` из конечных точек `/all` и `/flows` и добавьте их в тело запроса.

Добавление потока в проект перемещает поток из его предыдущего места. Поток не копируется.

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/projects/" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "name": "new_project_name",
  "description": "string",
  "components_list": [
```

```

    "3fa85f64-5717-4562-b3fc-2c963f66afa6"
  ],
  "flows_list": [
    "3fa85f64-5717-4562-b3fc-2c963f66afa6"
  ]
}'

```

34.3 Чтение проекта

Получите детали конкретного проекта.

Чтобы найти UUID вашего проекта, вызовите конечную точку чтения проектов.

```

curl -X GET \
  "$ASTRA_SERVER_URL/api/v1/projects/$PROJECT_ID" \
  -H "accept: application/json" \
  -H "x-api-key: $ASTRA_API_KEY"

```

Результат

```

{
  "name": "Starter Project",
  "description": "Manage your own projects. Download and upload projects.",
  "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
  "parent_id": null
}

```

34.4 Обновление проекта

Обновите информацию конкретного проекта с помощью запроса PATCH.

Каждый запрос PATCH обновляет проект значениями, которые вы отправляете. Обновляются только те поля, которые включены в ваш запрос. Если вы несколько раз отправляете одинаковые значения, обновление всё равно будет выполнено, даже если данные не изменились.

```

curl -X PATCH \
  "$ASTRA_SERVER_URL/api/v1/projects/b408ddb9-6266-4431-9be8-e04a62758331" \
  -H "accept: application/json" \
  -H "x-api-key: $ASTRA_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "name": "string",
    "description": "string",
    "parent_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",
    "components": [
      "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ],
    "flows": [
      "3fa85f64-5717-4562-b3fc-2c963f66afa6"
    ]
  }'

```

Результат

```

{
  "name": "string",
  "description": "string",
  "id": "b408ddb9-6266-4431-9be8-e04a62758331",
  "parent_id": null
}

```

34.5 Удаление проекта

Удалите конкретный проект.

```
curl -X DELETE \
"$ASTRA_SERVER_URL/api/v1/projects/$PROJECT_ID" \
-H "accept: */*" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

204 No Content

34.6 Экспорт проекта

Скачайте все потоки из проекта в виде ZIP-файла.

Флаг --output необязателен.

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/projects/download/$PROJECT_ID" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
--output astra-project.zip
```

34.7 Импорт проекта

Импортируйте проект и его потоки, загрузив ZIP-файл проекта Генеративный ИИ:

```
curl -X POST \
"$ASTRA_SERVER_URL/api/v1/projects/upload/" \
-H "accept: application/json" \
-H "Content-Type: multipart/form-data" \
-H "x-api-key: $ASTRA_API_KEY" \
-F "file=@20241230_135006_astra_flows.zip;type=application/zip"
```

35. Журналы

Получайте журналы для ваших потоков Генеративный ИИ и сервера.

35.1 Потокковая передача логи

Потокковая передача логи в реальном времени с использованием Server Sent Events (SSE):

```
curl -X GET \
  "$ASTRA_SERVER_URL/logs-stream" \
  -H "accept: text/event-stream" \
  -H "x-api-key: $ASTRA_API_KEY"
Результат
keepalive

{"1736355791151": "2025-01-08T12:03:11.151218-0500 DEBUG Building Chat Input\n"}

{"1736355791485": "2025-01-08T12:03:11.485380-0500 DEBUG consumed event add_message-153bcd5d-ef4d-4ece-8cc0-47c6b6a9ef92 (time in queue, 0.0000, client 0.0001)\n"}

{"1736355791499": "2025-01-08T12:03:11.499704-0500 DEBUG consumed event end_vertex-3d7125cd-7b8a-44eb-9113-ed5b785e3cf3 (time in queue, 0.0056, client 0.0047)\n"}

{"1736355791502": "2025-01-08T12:03:11.502510-0500 DEBUG consumed event end-40d0b363-5618-4a23-bbae-487cd0b9594d (time in queue, 0.0001, client 0.0004)\n"}

{"1736355791513": "2025-01-08T12:03:11.513097-0500 DEBUG Logged vertex build: 729ff2f8-6b01-48c8-9ad0-3743c2af9e8a\n"}

{"1736355791834": "2025-01-08T12:03:11.834982-0500 DEBUG Telemetry data sent successfully.\n"}

{"1736355791941": "2025-01-08T12:03:11.941840-0500 DEBUG Telemetry data sent successfully.\n"}

keepalive
```

35.2 Получить логи с необязательными параметрами

Получить логи с необязательными параметрами запроса:

- `lines_before`: Количество логов до метки времени или последнего лога.
- `lines_after`: Количество логов после метки времени.
- `timestamp`: Метка времени, с которой начинать получение логов.

Значения по умолчанию для всех трех параметров — 0. При значениях по умолчанию конечная точка возвращает последние 10 строк логов.

```
curl -X GET \
  "$ASTRA_SERVER_URL/logs?lines_before=0&lines_after=0&timestamp=0" \
  -H "accept: application/json" \
  -H "x-api-key: $ASTRA_API_KEY"
Результат
{
  "1736354770500": "2025-01-08T11:46:10.500363-0500 DEBUG Creating starter project Document Q&A\n",
  "1736354770511": "2025-01-08T11:46:10.511146-0500 DEBUG Creating starter project Image Sentiment Analysis\n",
  "1736354770521": "2025-01-08T11:46:10.521018-0500 DEBUG Creating starter project SEO Keyword Generator\n",
  "1736354770532": "2025-01-08T11:46:10.532677-0500 DEBUG Creating starter project Sequential Tasks Agents\n",
  "1736354770544": "2025-01-08T11:46:10.544010-0500 DEBUG Creating starter project Custom Component Generator\n",
  "1736354770555": "2025-01-08T11:46:10.555513-0500 DEBUG Creating starter project Prompt Chaining\n",
  "1736354770588": "2025-01-08T11:46:10.588105-0500 DEBUG Create service ServiceType.CHAT_SERVICE\n",
```



```
"1736354771021": "2025-01-08T11:46:11.021817-0500 DEBUG Telemetry data sent successfully.\n",  
"1736354775619": "2025-01-08T11:46:15.619545-0500 DEBUG Create service ServiceType.STORE_SERVICE\n",  
"1736354775699": "2025-01-08T11:46:15.699661-0500 DEBUG File 046-rocket.svg retrieved successfully from flow  
/Users/USER/Library/Caches/Генеративный ИИ/profile_pictures/Space.\n"  
}
```

36. Мониторинг

Конечные точки `/monitor` предназначены для внутренней функциональности Генеративный ИИ, в основном связанной с запуском потоков в **Playground**, хранением истории чата и генерацией журналов потоков.

Эта информация в первую очередь полезна тем, кто разрабатывает пользовательские компоненты или вносит вклад в кодовую базу Генеративный ИИ, требующую вызова или понимания этих конечных точек.

Для типичной разработки приложений с модулем «Генеративный ИИ» существуют более подходящие варианты мониторинга, отладки и управления памятью. Подробнее см. ниже:

- Логи: места хранения журналов Генеративный ИИ, параметры настройки и где просматривать журналы в визуальном редакторе
- Тестировать потоки в Playground: запуск потоков и просмотр истории сообщений
- Варианты управления памятью: варианты хранения данных Генеративный ИИ, включая базу данных, кеш и историю чата

36.1 Сборки Vertex

Конечные точки сборки Vertex (`/monitor/builds`) предназначены исключительно для функциональности **Playground**.

Когда вы запускаете поток в **Playground**, Генеративный ИИ вызывает конечную точку `/build/$FLOW_ID/flow` в файле `chat.py`. Этот вызов получает данные потока, создает граф и выполняет его. По мере выполнения каждого компонента (или узла) функция `build_vertex` вызывает `build_and_run`, которая может вызвать метод `def_build` отдельных компонентов, если он существует. Если у компонента нет функции `def_build`, сборка всё равно возвращает компонент.

Функция `build` позволяет компонентам выполнять логику во время выполнения. Например, компонент **Рекурсивный разделитель текста по символам** (https://github.com/langflow-ai/langflow/blob/main/src/lfx/src/lfx/components/langchain_utilities/recursive_character.py) является наследником класса `LCTextSplitterComponent`. Когда необходимо обработать текст, вызывается метод `build` родительского класса, который создаёт объект `RecursiveCharacterTextSplitter` и использует его для разбиения текста согласно заданным параметрам. Разбитый текст затем передаётся следующему компоненту. Всё это происходит при построении компонента.

Получить сборки Vertex

Получить сборки Vertex для конкретного потока.

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/monitor/builds?flow_id=$FLOW_ID" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "vertex_builds": {
    "ChatInput-NCmix": [
      {
        "data": {
          "results": {
```

```
"message": {
  "text_key": "text",
  "data": {
    "timestamp": "2024-12-23 19:10:57",
    "sender": "User",
    "sender_name": "User",
    "session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
    "text": "Hello",
    "files": [],
    "error": "False",
    "edit": "False",
    "properties": {
      "text_color": "",
      "background_color": "",
      "edited": "False",
      "source": {
        "id": "None",
        "display_name": "None",
        "source": "None"
      },
      "icon": "",
      "allow_markdown": "False",
      "positive_feedback": "None",
      "state": "complete",
      "targets": []
    },
    "category": "message",
    "content_blocks": [],
    "id": "c95bed34-f906-4aa6-84e4-68553f6db772",
    "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
  },
  "default_value": "",
  "text": "Hello",
  "sender": "User",
  "sender_name": "User",
  "files": [],
  "session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
  "timestamp": "2024-12-23 19:10:57+00:00",
  "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
  "error": "False",
  "edit": "False",
  "properties": {
    "text_color": "",
    "background_color": "",
    "edited": "False",
    "source": {
      "id": "None",
      "display_name": "None",
      "source": "None"
    },
    "icon": "",
    "allow_markdown": "False",
    "positive_feedback": "None",
    "state": "complete",
    "targets": []
  },
  "category": "message",
  "content_blocks": []
}
},
"outputs": {
  "message": {
    "message": {
```

```
"timestamp": "2024-12-23T19:10:57",
"sender": "User",
"sender_name": "User",
"session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
"text": "Hello",
"files": [],
"error": false,
"edit": false,
"properties": {
  "text_color": "",
  "background_color": "",
  "edited": false,
  "source": {
    "id": null,
    "display_name": null,
    "source": null
  },
  "icon": "",
  "allow_markdown": false,
  "positive_feedback": null,
  "state": "complete",
  "targets": []
},
"category": "message",
"content_blocks": [],
"id": "c95bed34-f906-4aa6-84e4-68553f6db772",
"flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
},
"type": "object"
}
},
"logs": { "message": [] },
"message": {
  "message": "Hello",
  "sender": "User",
  "sender_name": "User",
  "files": [],
  "type": "object"
},
"artifacts": {
  "message": "Hello",
  "sender": "User",
  "sender_name": "User",
  "files": [],
  "type": "object"
},
"timedelta": 0.015060124918818474,
"duration": "15 ms",
"used_frozen_result": false
},
"artifacts": {
  "message": "Hello",
  "sender": "User",
  "sender_name": "User",
  "files": [],
  "type": "object"
},
"params": "- Files: []\n Message: Hello\n Sender: User\n Sender Name: User\n Type: object\n",
"valid": true,
"build_id": "40aa200e-74db-4651-b698-f80301d2b26b",
"id": "ChatInput-NCmix",
"timestamp": "2024-12-23T19:10:58.772766Z",
"flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
```

```

    }
  ],
  "Prompt-BEn9c": [
    {
      "data": {
        "results": {},
        "outputs": {
          "prompt": {
            "message": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
            "type": "text"
          }
        }
      },
      "logs": { "prompt": [] },
      "message": {
        "prompt": {
          "repr": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
          "raw": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
          "type": "text"
        }
      },
      "artifacts": {
        "prompt": {
          "repr": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
          "raw": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
          "type": "text"
        }
      },
      "timedelta": 0.0057758750626817346,
      "duration": "6 ms",
      "used_frozen_result": false
    },
    {
      "artifacts": {
        "prompt": {
          "repr": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
          "raw": "Answer the user as if you were a GenAI expert, enthusiastic about helping them get started building something fresh.",
          "type": "text"
        }
      },
      "params": "None",
      "valid": true,
      "build_id": "39bbbfde-97fd-42a5-a9ed-d42a5c5d532b",
      "id": "Prompt-BEn9c",
      "timestamp": "2024-12-23T19:10:58.781019Z",
      "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
    }
  ],
  "OpenAIModel-7AjrN": [
    {
      "data": {
        "results": {},
        "outputs": {
          "text_output": {
            "message": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
            "type": "text"
          }
        }
      },

```

```
"model_output": { "message": "", "type": "unknown" }
},
"logs": { "text_output": [] },
"message": {
  "text_output": {
    "repr": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you
have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
    "raw": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you
have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
    "type": "text"
  }
},
"artifacts": {
  "text_output": {
    "repr": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you
have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
    "raw": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you
have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
    "type": "text"
  }
},
"timedelta": 1.034765167045407,
"duration": "1.03 seconds",
"used_frozen_result": false
},
"artifacts": {
  "text_output": {
    "repr": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have
in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
    "raw": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have
in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
    "type": "text"
  }
},
"params": "None",
"valid": true,
"build_id": "4f0ae730-a266-4d35-b89f-7b825c620a0f",
"id": "OpenAIModel-7AjrN",
"timestamp": "2024-12-23T19:10:58.790484Z",
"flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
},
],
"ChatOutput-sfUhT": [
{
  "data": {
    "results": {
      "message": {
        "text_key": "text",
        "data": {
          "timestamp": "2024-12-23 19:10:58",
          "sender": "Machine",
          "sender_name": "AI",
          "session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
          "text": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you
have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
          "files": [],
          "error": "False",
          "edit": "False",
          "properties": {
            "text_color": "",
            "background_color": "",
            "edited": "False",
            "source": {
```

```
      "id": "OpenAIModel-7AjrN",
      "display_name": "OpenAI",
      "source": "gpt-4o-mini"
    },
    "icon": "OpenAI",
    "allow_markdown": "False",
    "positive_feedback": "None",
    "state": "complete",
    "targets": []
  },
  "category": "message",
  "content_blocks": [],
  "id": "5688356d-9f30-40ca-9907-79a7a2fc16fd",
  "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
},
"default_value": "",
"text": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
"sender": "Machine",
"sender_name": "AI",
"files": [],
"session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
"timestamp": "2024-12-23 19:10:58+00:00",
"flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
"error": "False",
"edit": "False",
"properties": {
  "text_color": "",
  "background_color": "",
  "edited": "False",
  "source": {
    "id": "OpenAIModel-7AjrN",
    "display_name": "OpenAI",
    "source": "gpt-4o-mini"
  }
},
"icon": "OpenAI",
"allow_markdown": "False",
"positive_feedback": "None",
"state": "complete",
"targets": []
},
"category": "message",
"content_blocks": []
}
},
"outputs": {
  "message": {
    "message": {
      "timestamp": "2024-12-23T19:10:58",
      "sender": "Machine",
      "sender_name": "AI",
      "session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
      "text": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
      "files": [],
      "error": false,
      "edit": false,
      "properties": {
        "text_color": "",
        "background_color": "",
        "edited": false,
        "source": {
          "id": "OpenAIModel-7AjrN",
```

```

        "display_name": "OpenAI",
        "source": "gpt-4o-mini"
    },
    "icon": "OpenAI",
    "allow_markdown": false,
    "positive_feedback": null,
    "state": "complete",
    "targets": []
  },
  "category": "message",
  "content_blocks": [],
  "id": "5688356d-9f30-40ca-9907-79a7a2fc16fd",
  "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
},
"type": "object"
}
},
"logs": { "message": [] },
"message": {
  "message": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
  "sender": "Machine",
  "sender_name": "AI",
  "files": [],
  "type": "object"
},
"artifacts": {
  "message": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
  "sender": "Machine",
  "sender_name": "AI",
  "files": [],
  "type": "object"
},
"timedelta": 0.017838125000707805,
"duration": "18 ms",
"used_frozen_result": false
},
"artifacts": {
  "message": "Hello! I'm excited to help you get started on your journey to building something fresh! What do you have in mind? Whether it's a project, an idea, or a concept, let's dive in and make it happen!",
  "sender": "Machine",
  "sender_name": "AI",
  "files": [],
  "type": "object"
},
"params": "- Files: []\n Message: Hello! I'm excited to help you get started on your journey to building\n something fresh! What do you have in mind? Whether it's a project, an idea, or\n a concept, let's dive in and make it happen!\n Sender: Machine\n Sender Name: AI\n Type: object\n",
"valid": true,
"build_id": "1e8b908b-aba7-403b-9e9b-eca92bb78668",
"id": "ChatOutput-sfUdT",
"timestamp": "2024-12-23T19:10:58.813268Z",
"flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
}
]
}
}
}

```

Удалить сборки Vertex

Удалить сборки Vertex для конкретного потока.

```

curl -X DELETE \
"$ASTRA_SERVER_URL/api/v1/monitor/builds?flow_id=$FLOW_ID" \

```



```
-H "accept: */*" \
-H "x-api-key: $ASTRA_API_KEY"
Результат
204 No Content
```

36.2 Конечные точки сообщений

Конечные точки `/monitor/messages` сохраняют, возвращают, редактируют и удаляют записи в таблице сообщений в Генеративный ИИ.db. Обычно они вызываются не напрямую, а при запуске потоков, которые записывают историю сообщений, или при просмотре и изменении памяти в **Playground**.

Получить сообщения

Получить список всех сообщений:

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/monitor/messages" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Для фильтрации сообщений используйте параметры запроса `flow_id`, `session_id`, `sender` и `sender_name`.

Для сортировки результатов используйте параметр `order_by`.

В этом примере запрашиваются сообщения, отправленные Machine и AI в заданной чат-сессии (`session_id`), и сортируются по временной метке (`timestamp`).

```
curl -X GET \
"$ASTRA_SERVER_URL/api/v1/monitor/messages?flow_id=$FLOW_ID&session_id=01ce083d-748b-4b8d-97b6-33adbb6a528a&sender=Machine&sender_name=AI&order_by=timestamp" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
[
  {
    "id": "1c1d6134-9b8b-4079-931c-84dcaddf19ba",
    "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
    "timestamp": "2024-12-23 19:20:11 UTC",
    "sender": "Machine",
    "sender_name": "AI",
    "session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
    "text": "Hello! It's great to see you here! What exciting project or idea are you thinking about diving into today? Whether it's something fresh and innovative or a classic concept with a twist, I'm here to help you get started! Let's brainstorm together!",
    "files": "[]",
    "edit": false,
    "properties": {
      "text_color": "",
      "background_color": "",
      "edited": false,
      "source": {
        "id": "OpenAIModel-7AjrN",
        "display_name": "OpenAI",
        "source": "gpt-4o-mini"
      }
    },
    "icon": "OpenAI",
    "allow_markdown": false,
    "positive_feedback": null,
    "state": "complete",
    "targets": []
  },
  "category": "message",

```

```
"content_blocks": []
}
]
```

Удалить сообщения

Удалять отдельные сообщения по их идентификаторам.

В этом примере удаляется сообщение, полученное в предыдущем GET /messages примере.

```
curl -v -X DELETE \
"$ASTRA_SERVER_URL/api/v1/monitor/messages" \
-H "accept: */*" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d ["MESSAGE_ID_1", "MESSAGE_ID_2"]
```

Результат

```
204 No Content
```

Обновить сообщение

Обновить конкретное сообщение по его идентификатору.

В этом примере обновляется значение text в сообщении 3ab66cc6-c048-48f8-ab07-570f5af7b160.

```
curl -X PUT \
"$ASTRA_SERVER_URL/api/v1/monitor/messages/3ab66cc6-c048-48f8-ab07-570f5af7b160" \
-H "accept: application/json" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "text": "testing 1234"
}'
```

Результат

```
{
  "timestamp": "2024-12-23T18:49:06",
  "sender": "string",
  "sender_name": "string",
  "session_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
  "text": "testing 1234",
  "files": ["string"],
  "error": true,
  "edit": true,
  "properties": {
    "text_color": "string",
    "background_color": "string",
    "edited": false,
    "source": { "id": "string", "display_name": "string", "source": "string" },
    "icon": "string",
    "allow_markdown": false,
    "positive_feedback": true,
    "state": "complete",
    "targets": []
  },
  "category": "message",
  "content_blocks": [],
  "id": "3ab66cc6-c048-48f8-ab07-570f5af7b160",
  "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a"
}
```

Обновить идентификатор сессии

Обновить идентификатор сессии для сообщений.

В этом примере обновляется значение session_ID 01ce083d-748b-4b8d-97b6-33adbb6a528a на different_session_id.

```
curl -X PATCH \
"$ASTRA_SERVER_URL/api/v1/monitor/messages/session/01ce083d-748b-4b8d-97b6-
```

```
33adbb6a528a?new_session_id=different_session_id" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
[
  {
    "id": "8dd7f064-e63a-4773-b472-ca0475249dfd",
    "flow_id": "01ce083d-748b-4b8d-97b6-33adbb6a528a",
    "timestamp": "2024-12-23 18:49:55 UTC",
    "sender": "User",
    "sender_name": "User",
    "session_id": "different_session_id",
    "text": "message",
    "files": [],
    "edit": false,
    "properties": {
      "text_color": "",
      "background_color": "",
      "edited": false,
      "source": {
        "id": null,
        "display_name": null,
        "source": null
      },
    },
    "icon": "",
    "allow_markdown": false,
    "positive_feedback": null,
    "state": "complete",
    "targets": []
  },
  "category": "message",
  "content_blocks": []
}
```

Удалить сообщения по сессии

Удалить все сообщения для конкретной сессии.

```
curl -X DELETE \
"$ASTRA_SERVER_URL/api/v1/monitor/messages/session/different_session_id_2" \
-H "accept: */*" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

HTTP/1.1 204 No Content

36.3 Получить трассировки

Получение метаданных трассировки и деревьев соединений для конкретного потока.

Пример запроса

Используйте GET /monitor/traces и фильтр flow_id:

```
base_url = os.getenv("ASTRA_SERVER_URL", "http://localhost:7860")
api_key = os.getenv("ASTRA_API_KEY")
flow_id = "YOUR_FLOW_ID"

response = requests.get(
    f"{base_url}/api/v1/monitor/traces",
    params={"flow_id": flow_id, "page": 1, "size": 50},
    headers={"x-api-key": api_key, "accept": "application/json"},
    timeout=10,
)
```

```

response.raise_for_status()
traces = response.json()
print(traces)
const baseUrl = process.env.ASTRA_SERVER_URL ?? "http://localhost:7860";
const apiKey = process.env.ASTRA_API_KEY!;
const flowId = "YOUR_FLOW_ID";

async function listTraces() {
  const url = new URL("/api/v1/monitor/traces", baseUrl);
  url.searchParams.set("flow_id", flowId);
  url.searchParams.set("page", "1");
  url.searchParams.set("size", "50");

  const res = await fetch(url.toString(), {
    headers: {
      accept: "application/json",
      "x-api-key": apiKey,
    },
  });

  if (!res.ok) {
    throw new Error(`Request failed with status ${res.status}`);
  }

  const data = await res.json();
  console.log(data);
}

listTraces().catch(console.error);

curl -s "$ASTRA_SERVER_URL/api/v1/monitor/traces?flow_id=$FLOW_ID&page=1&size=50" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
| jq .

```

Пример ответа

```

{
  "traces": [
    {
      "id": "426656db-fc3c-4a3a-acf8-c60acf099543",
      "name": "Simple Agent - 9e774f60-857b-44b4-bbcd-87bd23848ee8",
      "status": "ok",
      "startTime": "2026-03-03T19:13:30.692628Z",
      "totalLatencyMs": 18693,
      "totalTokens": 2050,
      "flowId": "9e774f60-857b-44b4-bbcd-87bd23848ee8",
      "sessionId": "9e774f60-857b-44b4-bbcd-87bd23848ee8",
      "input": {
        "input_value": "Use tools to teach me about vertex graphs"
      },
      "output": {
        "message": {
          "text_key": "text",
          "data": {
            "timestamp": "2026-03-03 19:13:30 UTC",
            "sender": "Machine",
            "sender_name": "AI",
            "session_id": "9e774f60-857b-44b4-bbcd-87bd23848ee8",
            "text": "I can teach you the concept, but I couldn't pull the Wikipedia pages with the tool ... (truncated)"
          }
        }
      }
    }
  ]
}

```

```
  ],  
  "total": 1,  
  "pages": 1  
}
```

36.4 Получить транзакции

Получить все транзакции, представляющие собой взаимодействия между компонентами, для конкретного потока. Эта информация также доступна в flow logs.

```
curl -X GET \  
"$ASTRA_SERVER_URL/api/v1/monitor/transactions?flow_id=$FLOW_ID&page=1&size=50" \  
-H "accept: application/json" \  
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{  
  "items": [  
    {  
      "timestamp": "2024-12-23T20:05:01.061Z",  
      "vertex_id": "string",  
      "target_id": "string",  
      "inputs": {},  
      "outputs": {},  
      "status": "string",  
      "error": "string",  
      "flow_id": "3fa85f64-5717-4562-b3fc-2c963f66afa6",  
      "id": "3fa85f64-5717-4562-b3fc-2c963f66afa6"  
    }  
  ],  
  "total": 0,  
  "page": 1,  
  "size": 1,  
  "pages": 0  
}
```

37. Пользователи

Используйте конечную точку /users для управления учётными записями пользователей в модуле «Генеративный ИИ».

37.1 Добавить пользователя

Создайте новую учётную запись пользователя с указанным именем пользователя и паролем.

Требуется аутентификация как суперпользователь, если на сервере Генеративный ИИ включена аутентификация.

```
curl -X POST \
  "$ASTRA_URL/api/v1/users/" \
  -H "Content-Type: application/json" \
  -H "x-api-key: $ASTRA_API_KEY" \
  -d '{
    "username": "newuser2",
    "password": "securepassword123"
  }'
```

Запрос возвращает объект, описывающий нового пользователя. UUID пользователя хранится в user_id в базе данных Генеративный ИИ и возвращается как id в ответе API /users. Этот ключ user_id предназначен специально для управления пользователями Генеративный ИИ.

Результат

```
{
  "id": "10c1c6a2-ab8a-4748-8700-0e4832fd5ce8",
  "username": "newuser2",
  "profile_image": null,
  "store_api_key": null,
  "is_active": false,
  "is_superuser": false,
  "create_at": "2025-05-29T16:02:20.132436",
  "updated_at": "2025-05-29T16:02:20.132442",
  "last_login_at": null,
  "optins": {
    "github_starred": false,
    "dialog_dismissed": false,
    "discord_clicked": false
  }
}
```

37.2 Получить текущего пользователя

Получить информацию об аутентифицированном пользователе.

```
curl -X GET \
  "$ASTRA_URL/api/v1/users/whoami" \
  -H "accept: application/json" \
  -H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "id": "07e5b864-e367-4f52-b647-a48035ae7e5e",
  "username": "astra-admin",
  "profile_image": null,
  "store_api_key": null,
  "is_active": true,
}
```

```
"is_superuser": true,
"create_at": "2025-05-08T17:59:07.855965",
"updated_at": "2025-05-29T15:06:56.157860",
"last_login_at": "2025-05-29T15:06:56.157016"
}
```

37.3 Список всех пользователей

Получить постраничный список всех пользователей в системе.

Требуется аутентификация как суперпользователь, если на сервере Генеративный ИИ включена аутентификация.

```
curl -X GET \
"$ASTRA_URL/api/v1/users/?skip=0&limit=10" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "total_count": 3,
  "users": [
    {
      "id": "07e5b864-e367-4f52-b647-a48035ae7e5e",
      "username": "astra-admin",
      "profile_image": null,
      "store_api_key": null,
      "is_active": true,
      "is_superuser": true,
      "create_at": "2025-05-08T17:59:07.855965",
      "updated_at": "2025-05-29T15:06:56.157860",
      "last_login_at": "2025-05-29T15:06:56.157016",
      "optins": {
        "github_starred": false,
        "dialog_dismissed": true,
        "discord_clicked": false,
        "mcp_dialog_dismissed": true
      }
    },
    {
      "id": "c48a1f68-cc7e-491a-a507-a1a627708470",
      "username": "newuser",
      "profile_image": null,
      "store_api_key": null,
      "is_active": false,
      "is_superuser": false,
      "create_at": "2025-05-29T16:00:33.483386",
      "updated_at": "2025-05-29T16:00:33.483392",
      "last_login_at": null,
      "optins": {
        "github_starred": false,
        "dialog_dismissed": false,
        "discord_clicked": false
      }
    },
    {
      "id": "10c1c6a2-ab8a-4748-8700-0e4832fd5ce8",
      "username": "newuser2",
      "profile_image": null,
      "store_api_key": null,
      "is_active": false,
      "is_superuser": false,
      "create_at": "2025-05-29T16:02:20.132436",
      "updated_at": "2025-05-29T16:02:20.132442",

```

```

    "last_login_at": null,
    "optins": {
      "github_starred": false,
      "dialog_dismissed": false,
      "discord_clicked": false
    }
  }
]
}

```

37.4 Обновить пользователя

Изменить данные существующего пользователя с помощью PATCH-запроса.

Требуется аутентификация как суперпользователь, если на сервере Генеративный ИИ включена аутентификация.

В этом примере активируется учётная запись указанного пользователя и предоставляются права суперпользователя:

```

curl -X PATCH \
"$ASTRA_URL/api/v1/users/10c1c6a2-ab8a-4748-8700-0e4832fd5ce8" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "is_active": true,
  "is_superuser": true
}'

```

Результат

```

{
  "id": "10c1c6a2-ab8a-4748-8700-0e4832fd5ce8",
  "username": "newuser2",
  "profile_image": null,
  "store_api_key": null,
  "is_active": true,
  "is_superuser": true,
  "create_at": "2025-05-29T16:02:20.132436",
  "updated_at": "2025-05-29T16:19:03.514527Z",
  "last_login_at": null,
  "optins": {
    "github_starred": false,
    "dialog_dismissed": false,
    "discord_clicked": false
  }
}

```

37.5 Сбросить пароль

Изменить пароль указанного пользователя на новое значение.

Требуется аутентификация от имени целевого пользователя.

```

curl -X PATCH \
"$ASTRA_URL/api/v1/users/10c1c6a2-ab8a-4748-8700-0e4832fd5ce8/reset-password" \
-H "Content-Type: application/json" \
-H "x-api-key: $ASTRA_API_KEY" \
-d '{
  "password": "newsecurepassword123"
}'

```

Результат

```

{
  "id": "10c1c6a2-ab8a-4748-8700-0e4832fd5ce8",
  "username": "astra-admin",

```



```
"profile_image": null,
"store_api_key": null,
"is_active": true,
"is_superuser": true,
"create_at": "2025-05-08T17:59:07.855965",
"updated_at": "2025-05-29T15:06:56.157860",
"last_login_at": "2025-05-29T15:06:56.157016",
"optins": {
  "github_starred": false,
  "dialog_dismissed": true,
  "discord_clicked": false
}
}
```

37.6 Удалить пользователя

Удалить учётную запись пользователя из системы.

Требуется аутентификация как суперпользователь, если на сервере Генеративный ИИ включена аутентификация.

```
curl -X DELETE \
"$ASTRA_URL/api/v1/users/10c1c6a2-ab8a-4748-8700-0e4832fd5ce8" \
-H "accept: application/json" \
-H "x-api-key: $ASTRA_API_KEY"
```

Результат

```
{
  "detail": "User deleted"
}
```

38. Устранение неполадок

Эта страница содержит рекомендации по устранению неполадок, которые могут возникнуть при использовании Генеративный ИИ или при участии в его разработке.

38.1 Нет поля ввода в Playground

Если в **Playground** отсутствует поле ввода сообщения, убедитесь, что в вашем потоке есть компонент **Ввод в чат**, подключённый прямо или косвенно к порту **Ввод** компонента **Языковая модель** или **Агент**.

Поскольку **Playground** предназначен для потоков, использующих LLM в формате запрос–ответ, таких как чат-боты и агенты, поток должен содержать компоненты **Ввод в чат**, **Языковая модель/Агент** и **Вывод чата**, чтобы полностью поддерживаться чат-интерфейсом **Playground**.

Для получения дополнительной информации см. Тестирование потоков в Playground.

38.2 Отсутствует ключ, ключ не найден или неверный API-ключ

Если при запуске потока возникает ошибка API-ключа, попробуйте следующее:

- Для всех компонентов, требующих учётные данные, убедитесь, что в настройках этих компонентов указаны действительные учётные данные, например в поле **Ключ API**.
- Если вы храните учётные данные в глобальных переменных Генеративный ИИ, убедитесь, что выбрана правильная глобальная переменная и что она содержит действительные учётные данные.
- Убедитесь, что предоставленные учётные данные активны, имеют необходимые разрешения и при необходимости — достаточный баланс на счёте для выполнения требуемого действия. Например, поставщики моделей требуют кредиты для использования их LLM.

38.3 Проблемы MCP в модуле «Генеративный ИИ»

Ниже описаны проблемы, которые могут возникнуть при использовании Генеративный ИИ в качестве сервера или клиента MCP.

Сервер MCP проекта по умолчанию работает только без аутентификации

Если сервер MCP проекта по умолчанию работает без аутентификации, но перестает работать после добавления ключа API в конфигурацию, возможно, ключ API добавлен в неправильный раздел конфигурации.

Сервер MCP по умолчанию использует потоковый HTTP-транспорт. Ключ API необходимо добавить в массив `args`, который передаётся в `mcp-proxy`, а не в объект `env`.

Массив `args` должен включать `--headers`, `"x-api-key"` и значение вашего ключа. Например:

```
{
  "mcpServers": {
    "PROJECT_NAME": {
      "command": "uvx",
      "args": [
        "mcp-proxy",
        "--transport",
        "streamablehttp",

```

```

"--headers",
"x-api-key",
"YOUR_API_KEY",
"http://SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable"
]
}
}
}

```

Для получения дополнительной информации см. Подключение клиентов к серверу MCP Генеративный ИИ.

Claude for Desktop не использует инструменты сервера MCP корректно

Если Claude for Desktop не использует инструменты вашего сервера корректно, попробуйте явно указать путь к исполняемому файлу uvx или prx в файле конфигурации `claude_desktop_config.json`:

1. Чтобы найти путь к uvx, выполните `which uvx`. Чтобы найти путь к prx, выполните `which prx`.
2. В файле `claude_desktop_config.json` добавьте путь к конфигурации сервера MCP Генеративный ИИ, как показано ниже.

```

{
  "mcpServers": {
    "PROJECT_NAME": {
      "command": "PATH_TO_UVX",
      "args": [
        "mcp-proxy",
        "http://SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable"
      ]
    }
  }
}
{
  "mcpServers": {
    "PROJECT_NAME": {
      "command": "PATH_TO_NPX",
      "args": [
        "-y",
        "supergateway",
        "--sse",
        "http://SERVER_ADDRESS/api/v1/mcp/project/PROJECT_ID/streamable"
      ]
    }
  }
}

```

Потоки MCP на основе браузера не открывают браузер в Windows

Это известная проблема при использовании инструментов MCP с действиями навигации в браузере, такими как Playwright, в Windows: агент может успешно вызвать инструмент, но вкладка или окно браузера не открываются.

Проблема возникает из-за того, что сервер MCP запускается из процесса Python, что препятствует открытию окон браузера в WSL или Windows.

Чтобы обойти эту проблему, используйте подход с автономным сервером MCP, описанный в репозитории Playwright MCP. После запуска сервера добавьте его как HTTP/SSE-сервер в модуле «Генеративный ИИ».

Для других инструментов навигации в браузере обратитесь к документации поставщика за конкретными рекомендациями по устранению неполадок.

«Нет подключённых инструментов или подсказок» на серверах MCP в смешанных средах Windows/WSL

Если при использовании Генеративный ИИ Desktop в качестве сервера MCP с клиентами в разных средах (например, Генеративный ИИ на хосте Windows и клиент MCP в WSL) возникают ошибки «Нет подключённых инструментов или подсказок» или сбой соединения, это связано с изоляцией сети между Windows и WSL.

WSL не может напрямую обращаться к сервисам на localhost Windows. Генеративный ИИ, запущенный на хосте Windows, недоступен клиентам WSL по адресу localhost:7860.

Чтобы обойти эту проблему, запустите сервер и клиент в одной операционной среде.

Либо настройте Генеративный ИИ Desktop принимать соединения из WSL по IP-адресу Windows по умолчанию 10.255.255.254:7860 вместо localhost.

Компонент «Инструменты MCP» теряет настройку «Режим инструмента» после обновления потоков

Если вы обновляете существующий поток, использующий компонент **Инструменты MCP в Режиме инструмента**, компонент может потерять эту настройку после обновления. Это может нарушить работу потоков, которые полагаются на **Режим инструмента** для предоставления инструментов MCP агентам.

Если вы столкнулись с этим при обновлении потока, созданного в модуле «Генеративный ИИ» версии 1.7.1 или более ранней, выполните следующие действия:

1. Выберите компонент **Инструменты MCP** в вашем потоке.
2. Нажмите **Код**, чтобы открыть редактор кода компонента.
3. В структуре шаблона компонента найдите поле `tool_mode` в разделе `inputs`. Это поле может отсутствовать или иметь значение `false`.
4. Добавьте поле `tool_mode`, если оно отсутствует, и установите его значение `true`:
`tool_mode = True`
5. Нажмите **Проверить и сохранить**, чтобы применить изменения.

38.4 Ошибки ограничения длины токенов в компонентах модели эмбедингов

Ошибки, связанные с длиной токенов, могут возникать, если стратегия разбиения на фрагменты не соответствует ограничениям токенизации используемой модели эмбедингов. Для получения дополнительной информации см. Ошибки токенизации из-за размера фрагмента.

ЧАСТЬ 2. МОДУЛЬ «МАШИННОЕ ОБУЧЕНИЕ»

39. Введение

39.1. Область применения

Программное обеспечение Машинное обучение (далее – Платформа, платформа Машинное обучение) обрабатывает структурированные и неструктурированные массивы данных, обучает модели ИИ (искусственного интеллекта) для создания баз знаний, предиктивной аналитики в промышленности и сфере финансов и здравоохранения. Платформа предоставляет все необходимые инструменты и ресурсы для выполнения полного цикла работы в области data science.

40. Руководство администратора платформы Астра ИИ

40.1 Начало работы

Базовые шаги для освоения платформы:

- Регистрация и Авторизация
- Обзор интерфейса и работа с файлами
- Личный кабинет пользователя
- Демо-тур

40.2 Проекты

Проекты — это центральное рабочее пространство для решения ваших задач. Из этого руководства вы узнаете, как:

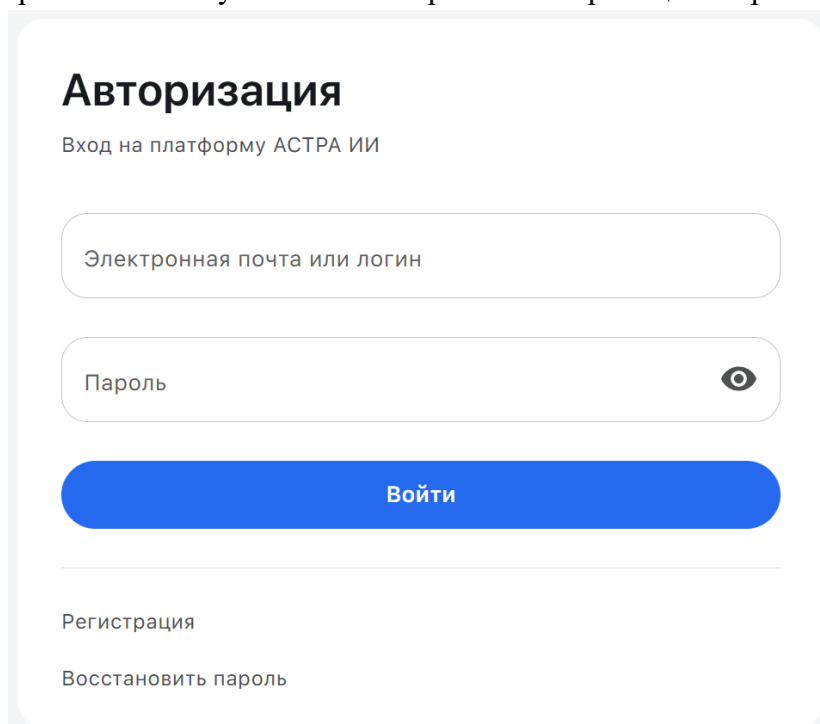
- Создавать проекты
- Работать с датасетами
- Обучать модели ИИ
- Использовать модели ИИ
- Настраивать коннекторы
- Добавлять дашборды

41. Начало работы с платформой

Для доступа к платформе АСТРА ИИ вам потребуется учетная запись.

41.1 Авторизация

При запуске приложения по умолчанию открывается страница авторизации:



Авторизация

Вход на платформу АСТРА ИИ

Электронная почта или логин

Пароль 

Войти

[Регистрация](#)

[Восстановить пароль](#)

1. Введите ваши данные в поля:

- Электронная почта или логин
- Пароль

2. Нажмите «Войти»

Если у вас нет аккаунта, нажмите на ссылку «Регистрация». Если вы забыли пароль, воспользуйтесь ссылкой «Восстановить пароль»

41.2 Регистрация

В случае отсутствия учетной записи необходимо пройти процесс регистрации.

Регистрация

Регистрация на платформе АСТРА ИИ

Электронная почта

Логин

Имя

Фамилия

Пароль

Подтвердите пароль

Зарегистрироваться

[Авторизация](#)

1. Перейдите по соответствующей ссылке на странице авторизации.
2. Заполните поля формы:
 - Электронная почта — будет использоваться для уведомлений и восстановления доступа.
 - Логин — ваше уникальное имя на платформе.
 - Имя и Фамилия.
 - Пароль и Подтвердите пароль
3. Нажмите «Зарегистрироваться».

Вы можете сохранить связку логин и пароль в браузере для автоматического входа на текущем устройстве.

Для возврата на страницу входа можно использовать ссылку «Авторизация» внизу формы.

42. Обзор интерфейса

Этот раздел поможет вам освоиться с основными элементами управления платформы АСТРА ИИ.

42.1 Файловое хранилище

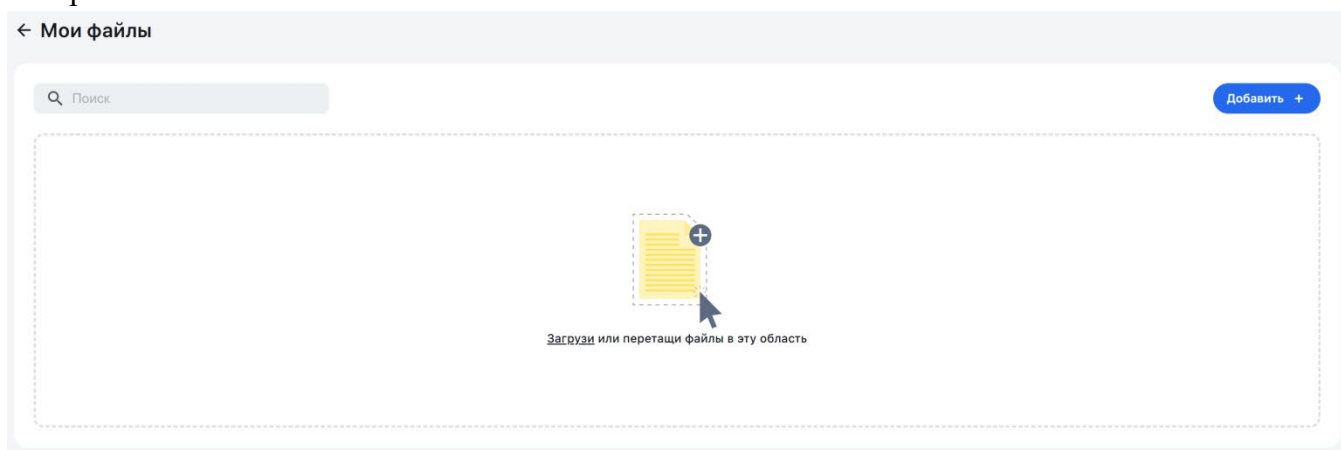
«**Мои файлы**» — это ваше персональное хранилище данных на платформе.

Чтобы открыть файловое хранилище, нажмите на иконку папки  в правом верхнем углу.



На открывшейся странице отображается информация о файлах/папках.

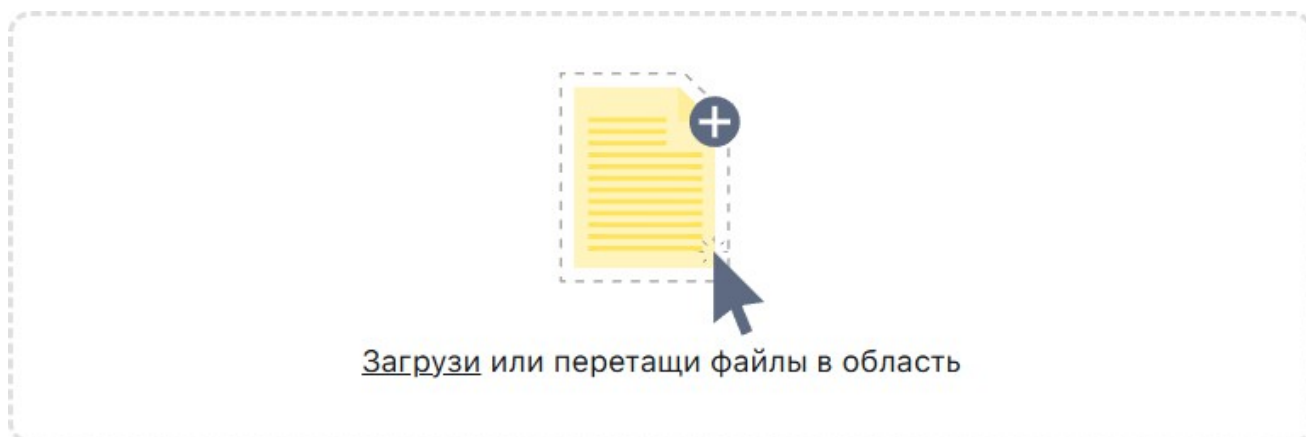
Если вы открыли этот раздел впервые и еще не загружали файлы в систему, страница отображается так:



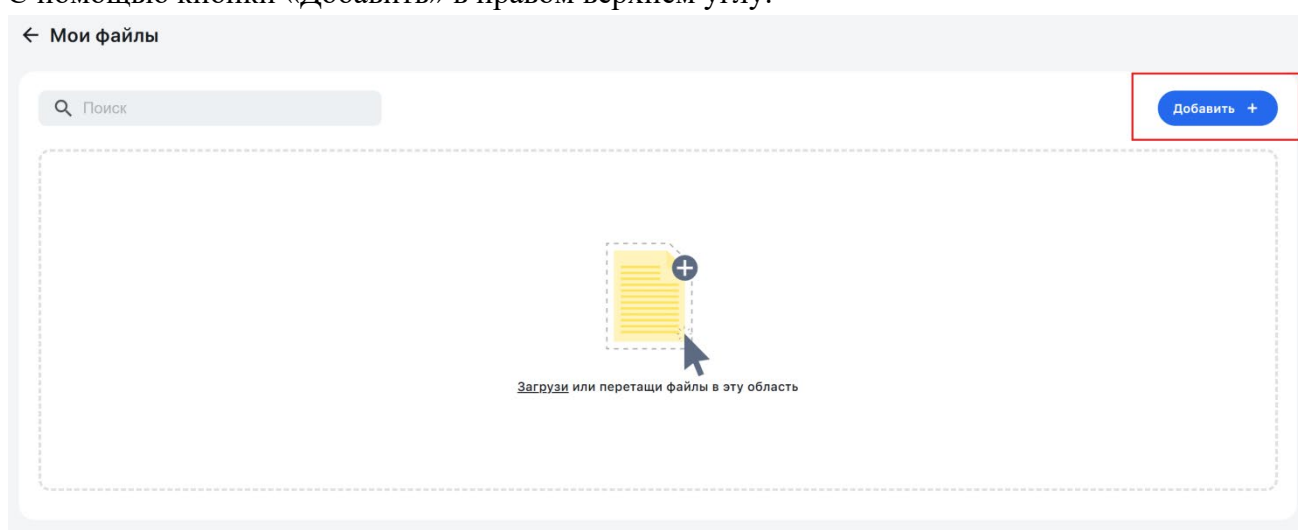
42.1.1 Способы добавления данных

Чтобы использовать данные на платформе, необходимо предварительно их загрузить в файловое хранилище. Это можно сделать двумя способами:

1. Перетаскивание (Drag & Drop) . Если хранилище пустое, просто перетащите нужные файлы или папки с вашего компьютера в специальную область.

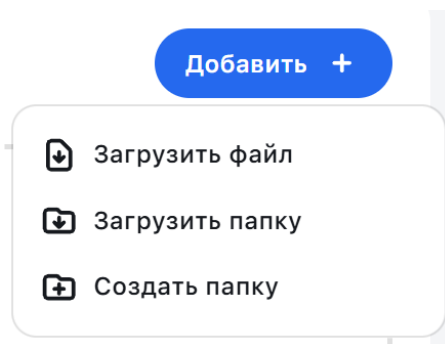


2. С помощью кнопки «Добавить» в правом верхнем углу.



В выпадающем меню выберите один из вариантов:

- «Загрузить» — для добавления отдельных файлов.
- «Создать папку» — если вы хотите собрать файлы на платформе в одну папку для удобства и далее в нее будете загружать файлы.
- «Загрузить папку» — если на компьютере у вас уже есть папка с данными, можете ее сразу загрузит на платформу.



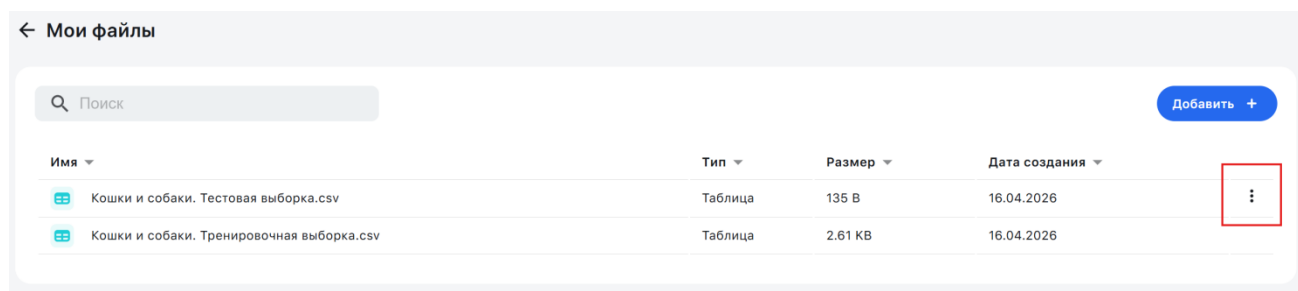
На платформе Машинное обучение поддерживаются следующие форматы:

- Табличные данные: .csv , .xlsx .
- Изображения : .png , .jpeg , .jpg .
- Классы изображений : папки с изображениями, где название каждой папки является названием класса.
- Текстовые файлы. можно загружать текстовые файлы в формате .txt .

42.1.2 Управление файлами и папками

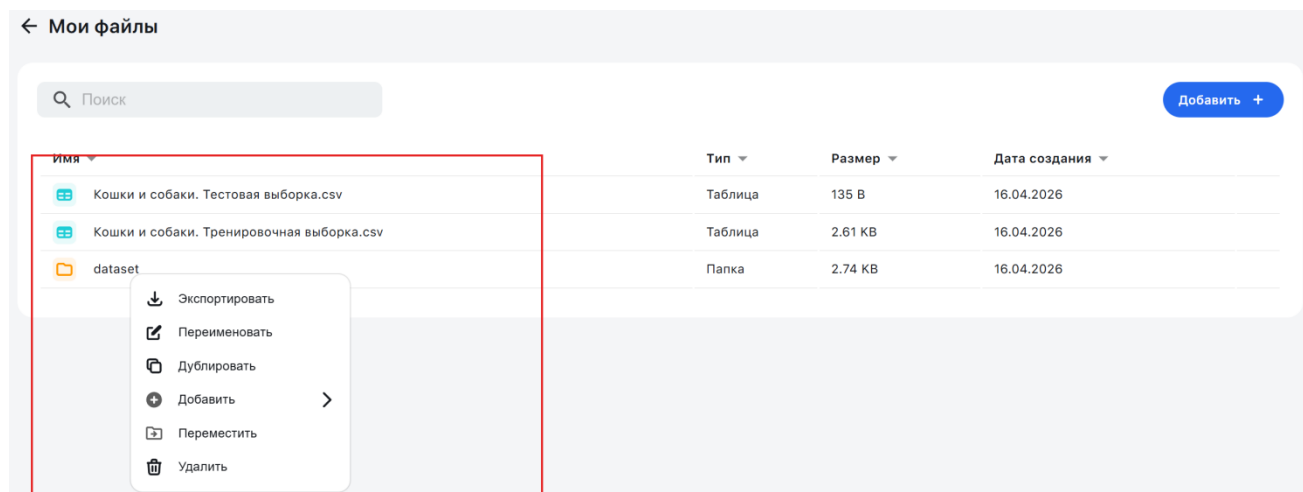
Наведите курсор на любой файл или папку, чтобы справа появились иконки действий:

- «Редактировать» — изменить название.
- «Скачать» — экспортировать файл/папку на компьютер.
- «Удалить» — безвозвратно удалить элемент.



Действия с папкой

Нажмите на любую папку правой кнопкой мыши (ПКМ), чтобы открыть контекстное меню. Это позволит вам загрузить файлы/папки или создать новую папку внутри выбранной.

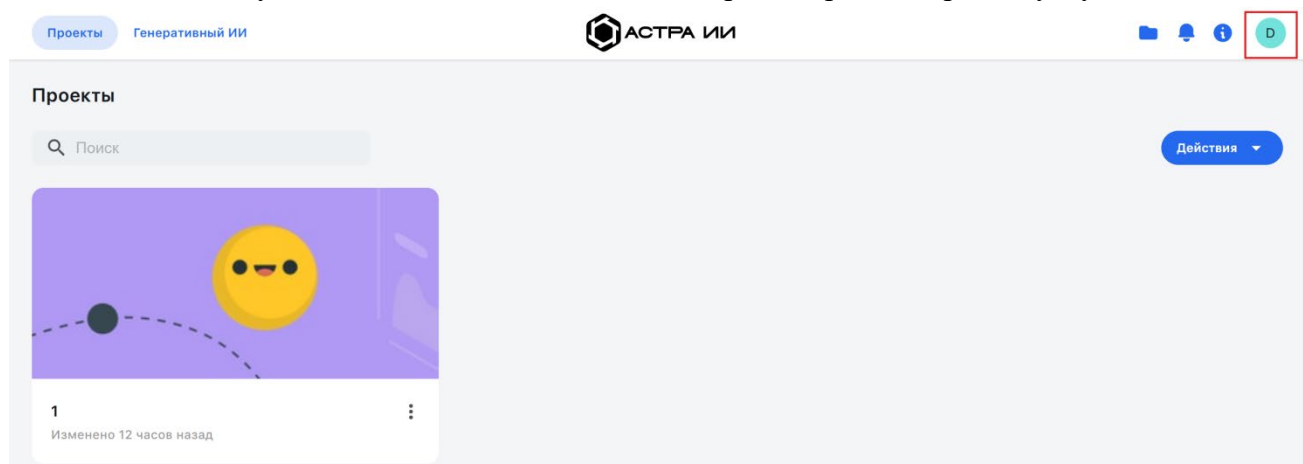


42.2 Уведомления

В этом разделе отображаются важные события, связанные с вашей активностью на платформе. При изменении рейтинга, появлении новых достижений или обновлении условий использования платформы, а также при добавлении новых возможностей вы будете получать уведомления. Они отображаются на верхней панели в правом верхнем углу экрана. Для просмотра нажмите кнопку звоночка 🔔.

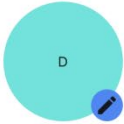
42.3 Личный кабинет

Личный кабинет — это персональная страница для управления профилем. Чтобы войти, нажмите на иконку с вашими инициалами или аватаром в правом верхнем углу:



После этого откроется страница профиля:

← Профиль



User Default
test@test.ru

Личные данные

Безопасность

Выйти

Основная информация

Фамилия *
User

Имя *
Default

Отчество

Логин
test@test.ru

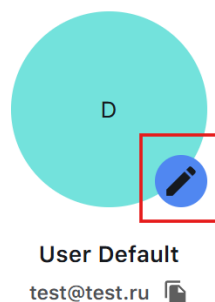
Номер телефона

Сохранить изменения

42.3.1 Личные данные

Эта вкладка открывается по умолчанию. Здесь вы можете управлять основной информацией о себе: изменить свои Фамилию, Имя, Отчество и Номер телефона

Чтобы изменить аватарку, нажмите на кнопку редактирования, а затем выберите желаемое изображение:



Убедитесь, что после окончания загрузки выбранное изображение будет установлено в качестве аватара.

После внесения правок не забудьте нажать «Сохранить изменения»:

Основная информация

Фамилия *
User

Имя *
Default

Отчество
Иванович

Логин
test@test.ru

Номер телефона

Сохранить изменения

Отменить

42.3.2 Безопасность

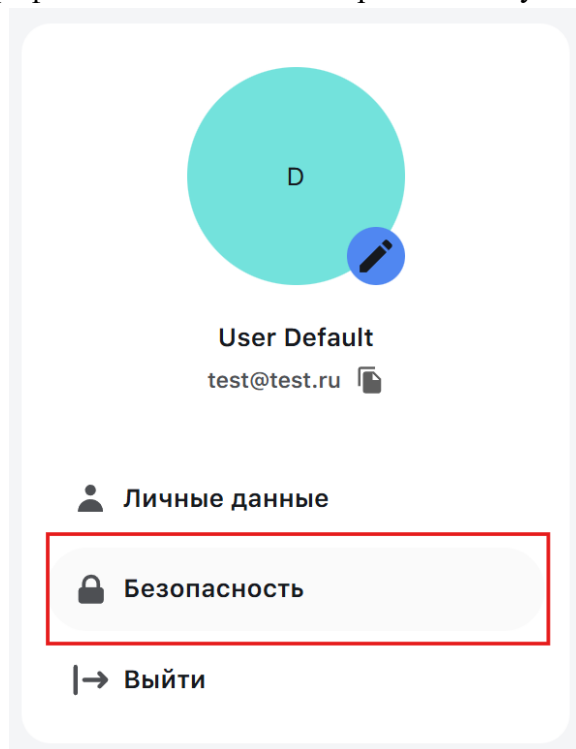
На этой вкладке вы можете:

- Изменить основную почту.
- Изменить пароль для входа на платформу.

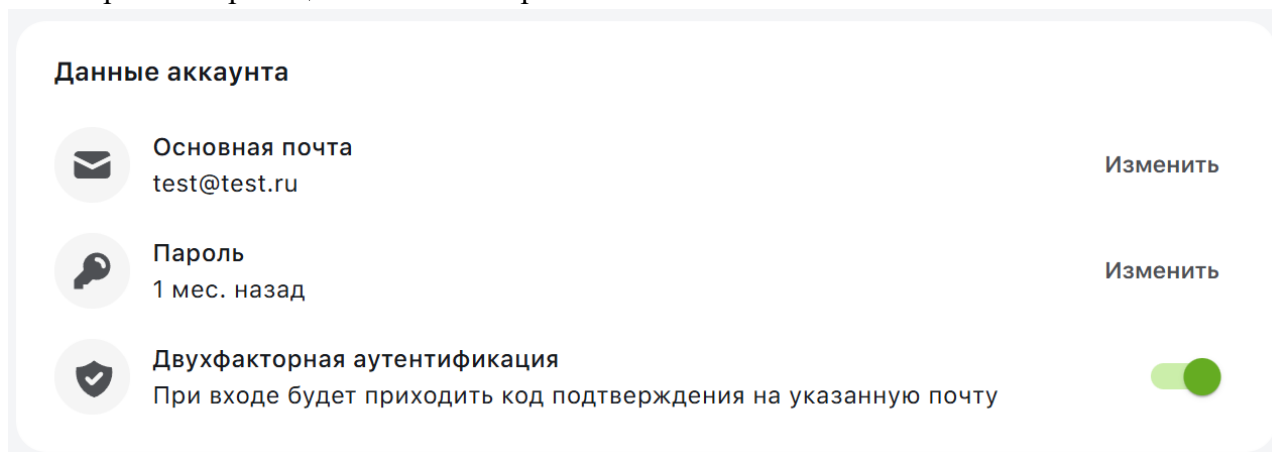
— Установить или снять двухфакторную аутентификацию для безопасности аккаунта.

Чтобы изменить параметры авторизации, выполните следующее:

1. На странице профиля пользователя выберите вкладку Безопасность :



Откроется страница с такими настройками:



Напротив поля «Основная почта» нажмите «Изменить» и задайте новый адрес.

Обратите внимание, что на указанную электронную почту и номер телефона будут приходить уведомления от платформы АСТРА ИИ.

Напротив поля «Пароль» нажмите «Изменить» и задайте новый пароль. Убедитесь, что выполняется вход в Систему с использованием нового пароля.

Напротив поля «Двухфакторная аутентификация» при необходимости установите переключатель. В результате откроется окно для ввода кода подтверждения:

**Подтверждение включения двухфакторной
аутентификации**

Мы отправили письмо код на ede*****.com

Продолжить

Скопируйте код в письме на электронной почте и введите в ячейки. Нажмите на кнопку «Продолжить».

Убедитесь, что выполняется вход в Систему с использованием нового пароля.

43. Управление проектами

Проект — это ваше основное рабочее пространство на платформе. В этом разделе вы узнаете, как создавать новые проекты для своих задач и управлять уже существующими.

43.1 Как создать проект

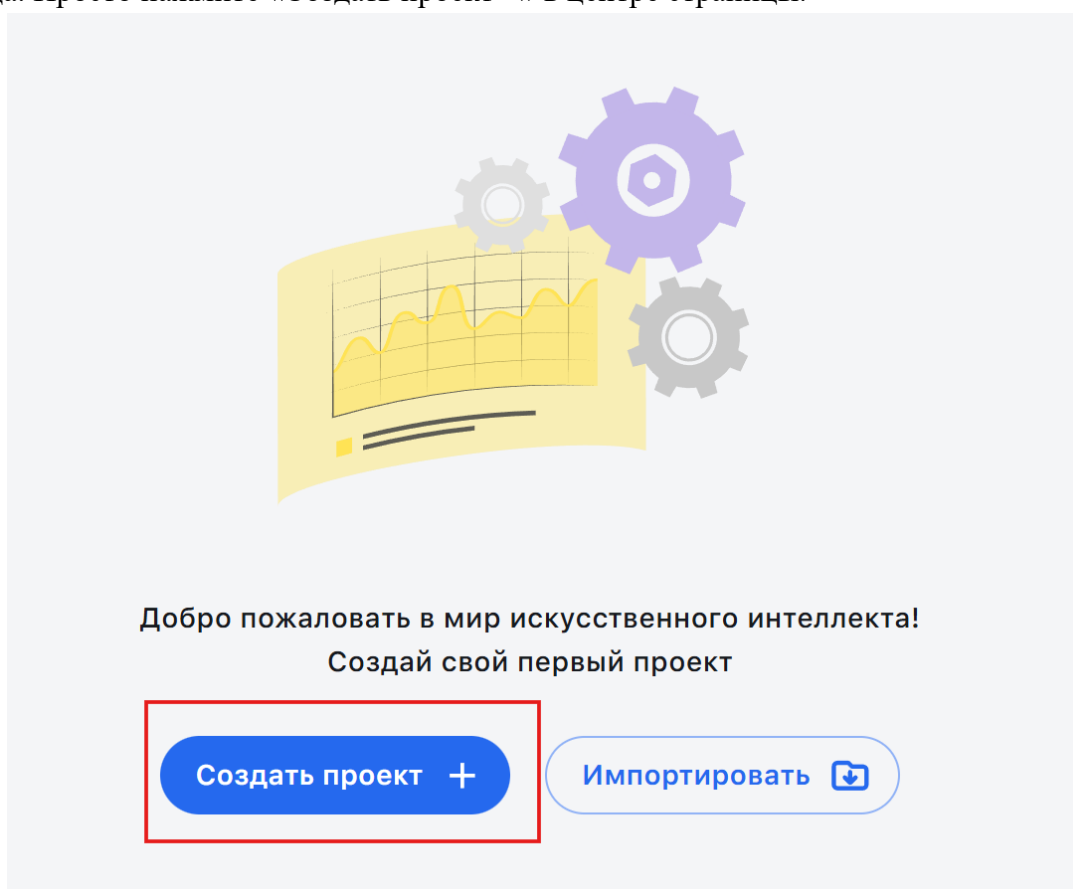
Шаг 1: Перейти в раздел «Проекты»

Для начала работы откройте раздел «Проекты» в верхнем левом углу:



Шаг 2: Добавить новый проект

Если вы новый пользователь и у вас нет еще никаких проектов, откроется пустая стартовая страница. Просто нажмите «Создать проект +» в центре страницы.



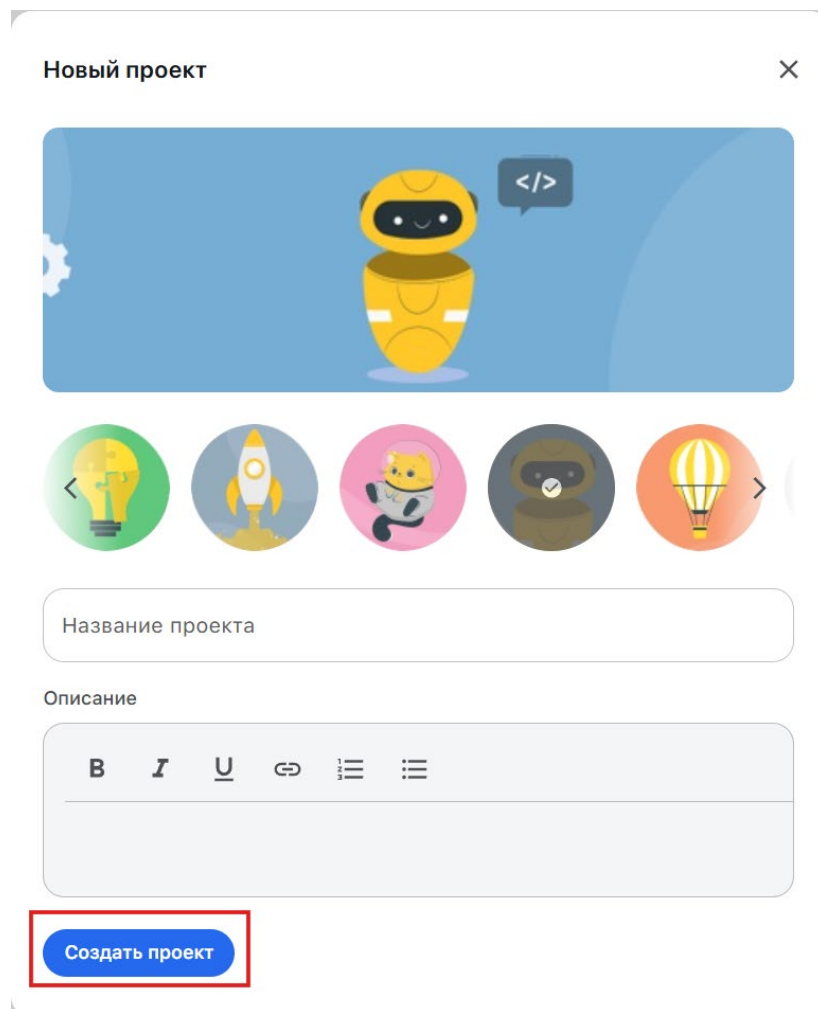
Если у вас уже есть созданные проекты, тогда откроется страница с проектами. На странице со списком ваших проектов нажмите на синюю кнопку «Действия» в правом верхнем углу. В выпадающем меню выберите пункт «Создать»

Шаг 3: Заполнить информацию о проекте

В открывшейся форме «Новый проект» заполните поля:

- Название проекта
- Описание (необязательно)

Затем нажмите кнопку «Создать проект».



Новый проект

Название проекта

Описание

Создать проект

Готово! Вы будете автоматически перенаправлены в карточку только что созданного проекта, где сможете продолжить работу: загружать данные, строить пайплайны и обучать модели.

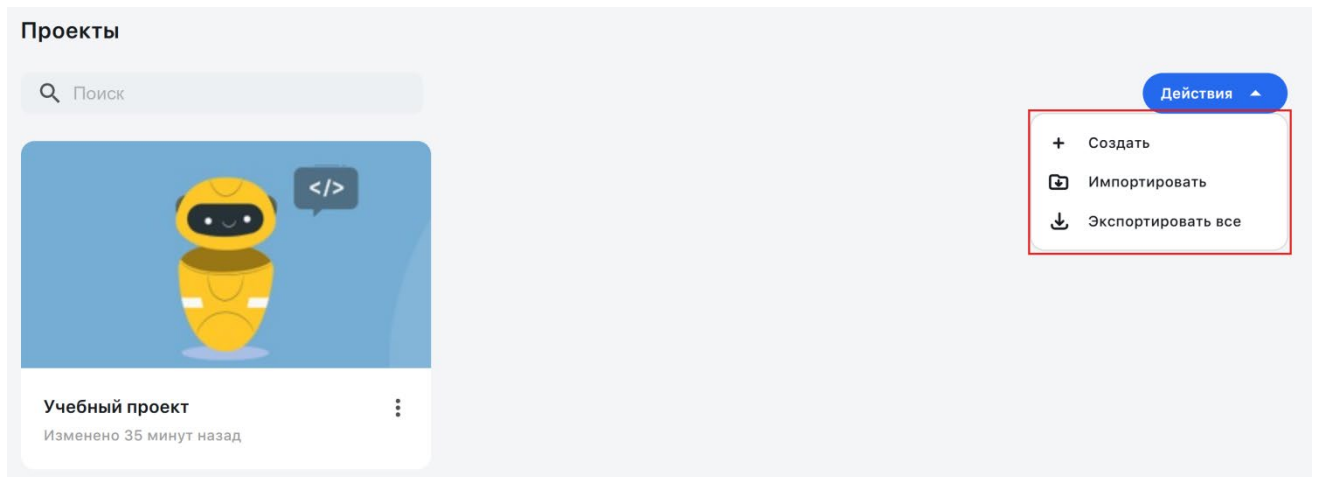
43.2 Работа с существующими проектами

На странице со списком всех проектов доступны следующие функции:

1. Общие действия с проектами

Кнопка «Действия» — это меню в правом верхнем углу, позволяет работать со всеми проектами сразу.

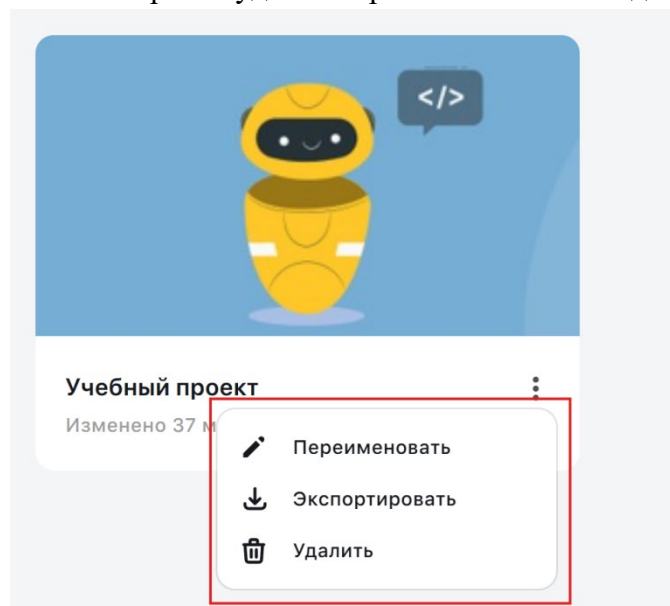
- Создать — открыть форму создания нового проекта.
- Импортировать — загрузить проект из ранее экспортированного .zip архива
- Экспортировать все — скачать все проекты одним архивом.



2. Действия с конкретным проектом

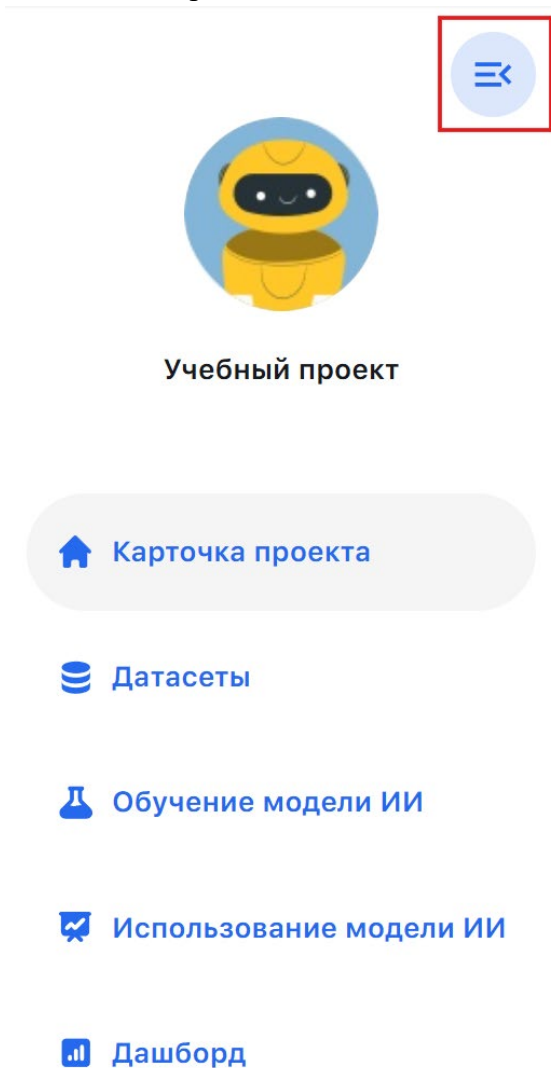
У каждой карточки проекта в правом нижнем углу есть меню в виде трёх точек. Нажав на него, вы можете:

- Переименовать — изменить название проекта.
- Экспортировать — скачать выбранный проект в виде .zip архива.
- Удалить — безвозвратно удалить проект со всеми его данными.



44. Меню проекта

После открытия проекта вы по умолчанию попадаете в раздел «Карточка проекта». Это главная страница, которая служит центром управления для всех его компонентов. Для навигации по другим инструментам проекта используется боковое меню, расположенное в левой части экрана. Оно включает разделы: Карточка проекта, Датасеты, Обучение модели ИИ, Использование модели ИИ и Дашборд



Примечание :

Боковое меню можно свернуть, чтобы освободить больше места на экране для основной рабочей области. Для этого нажмите на иконку , расположенную в верхней части меню. Чтобы развернуть меню обратно, нажмите на нее еще раз.

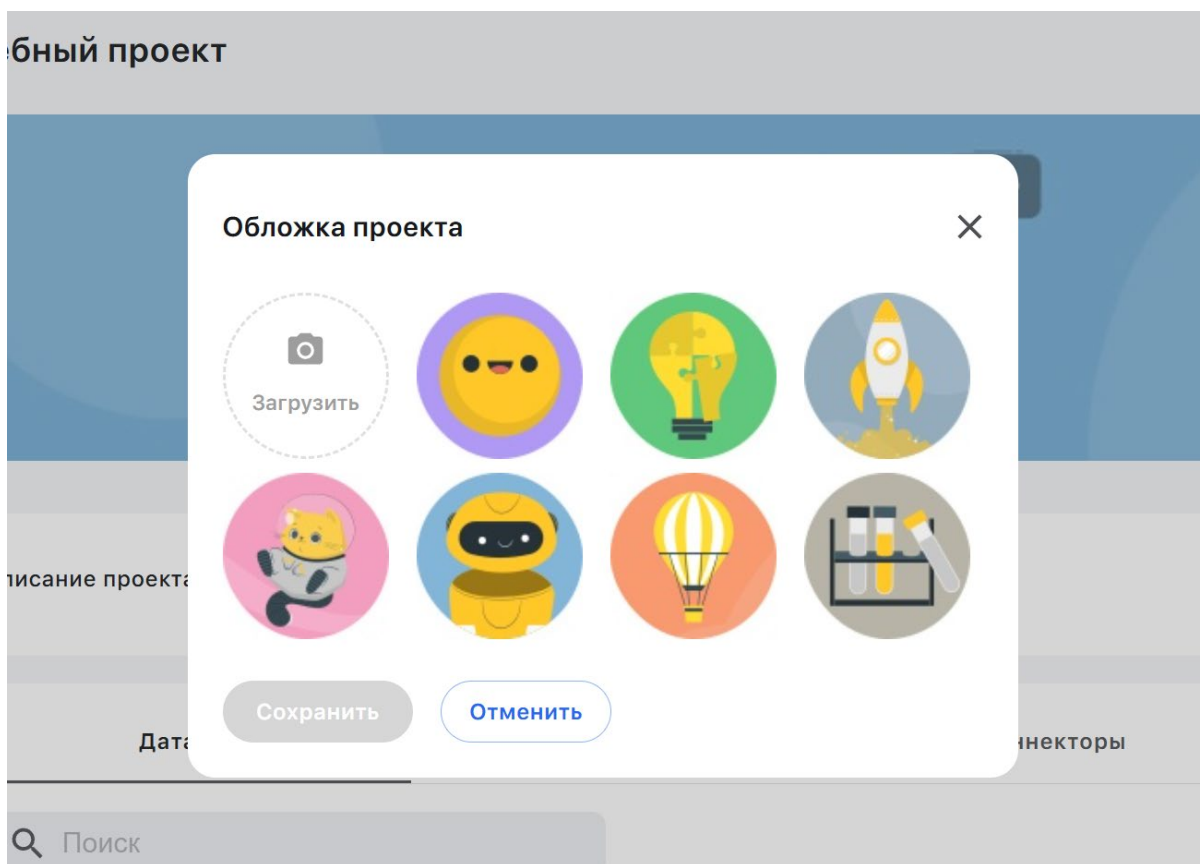
В таблице ниже кратко описано назначение каждого раздела.

Раздел	Описание
Карточка проекта	Основной раздел проекта, содержащий общую информацию (описание, обложка) и обеспечивающий доступ ко всем его сущностям: датасетам, моделям и т.д.
Датасеты	Раздел для управления датасетами, которые были созданы в проекте. Подробнее о создании датасетов вы можете посмотреть в разделе Датасеты.
Обучение модели ИИ	Раздел для визуального проектирования пайплайнов (блок-схем) с целью обучения модели.
Использование модели ИИ	Раздел для построения пайплайнов, применяющих обученную модель к новым данным для решения ее целевых задач.
Дашборд	Инструмент добавления интерактивных панелей с графиками и таблицами для наглядного представления исходных данных, результатов работы моделей и ключевых метрик.

45. Карточка проекта

Карточка проекта — это главная страница, которая содержит всю основную информацию о проекте и предоставляет быстрый доступ ко всем объектам: датасетам, моделям, отчетам, дашбордам и коннекторам.

В верхней части страницы отображаются название и описание проекта. Вы можете персонализировать свой проект, установив для него уникальную обложку. Чтобы изменить ее, наведите курсор на изображение. В правом нижнем углу появится кнопка «Изменить обложку». Нажмите на нее и выберите подходящее изображение из предложенных или загрузите свое.

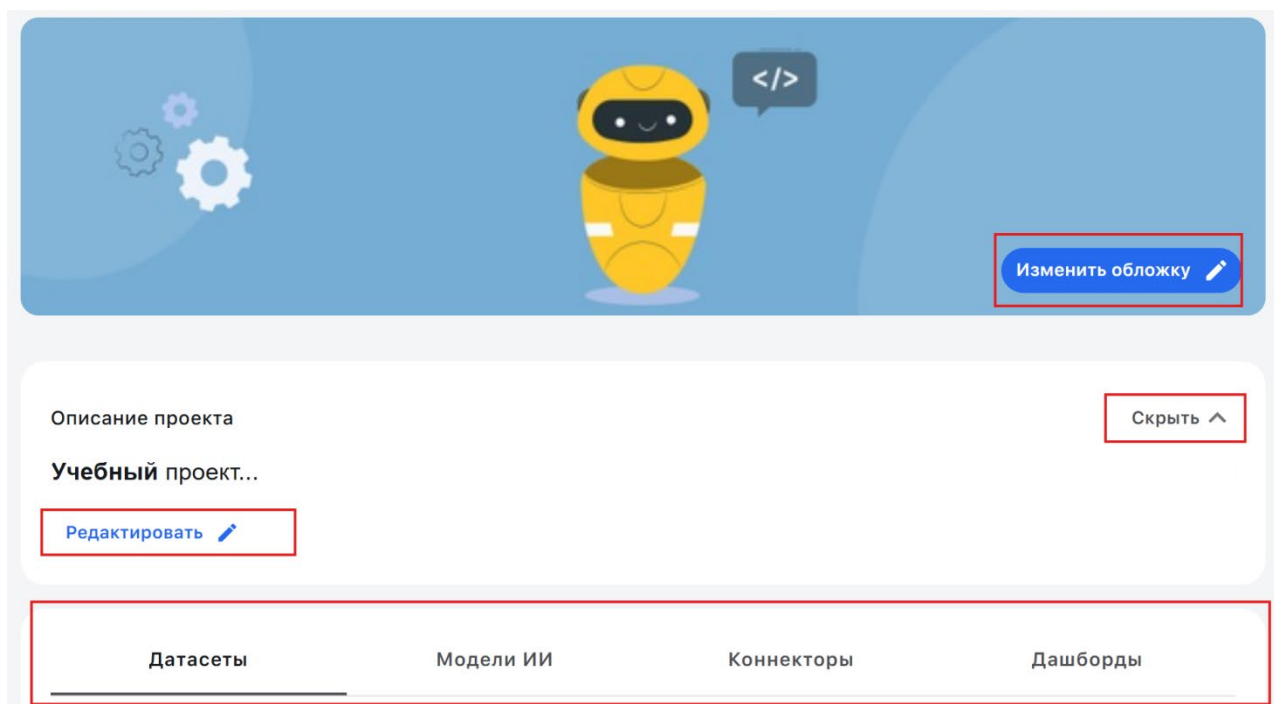


Блок с описанием по умолчанию скрыт. Чтобы развернуть его, нажмите на заголовок «Описание проекта» или кнопку «Показать». Чтобы свернуть его обратно, нажмите «Скрыть». Для внесения правок в описание:

1. Нажмите «Редактировать».
2. В текстовом редакторе измените текст, используя доступные инструменты форматирования (жирный, курсив, нижнее подчеркивание, списки).

По завершении нажмите «Сохранить изменения».

В описании обычно указывают цель проекта, ключевые задачи, основные этапы или краткое содержание проделанной работы.



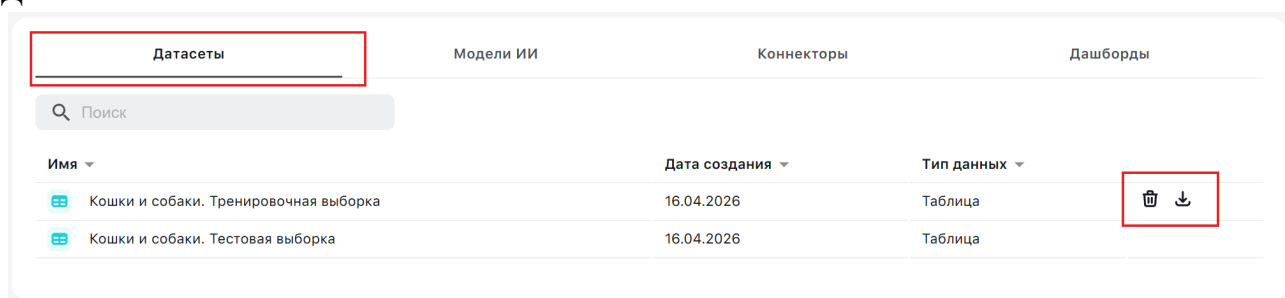
Под описанием расположена панель с вкладками, где собраны все сущности, созданные или использованные в рамках проекта.

45.1 Датасеты

На вкладке «Датасеты» автоматически отображаются все датасеты, которые были использованы на рабочих областях при построении пайплайнов для обучения и применения моделей. Каждый элемент в списке кликабелен и ведет на страницу соответствующего датасета. Для каждого датасета доступны действия:

- Скачать (↓)
- Удалить (🗑)

Подробнее о загрузке и управлении всеми вашими наборами данных читайте в разделе Датасеты.



45.2 Модели ИИ

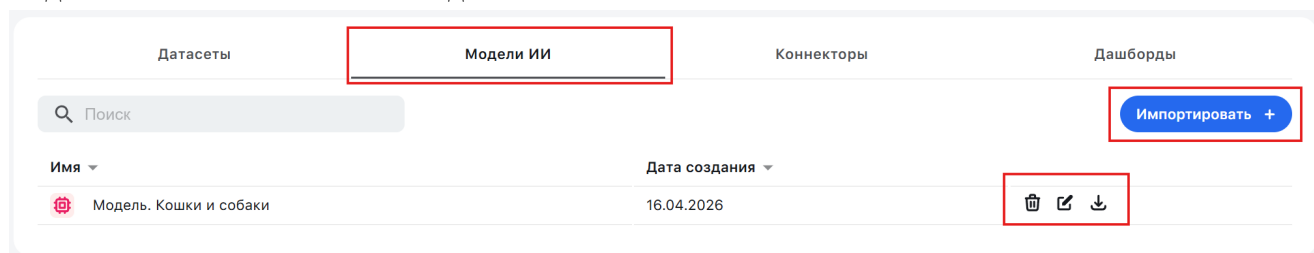
На этой вкладке находится список всех моделей ИИ, которые были обучены, сохранены или импортированы в проект. Если моделей еще нет, вы увидите предложение импортировать их.

Вы можете сохранять модели после обучения на рабочих областях (тогда они автоматически будут добавлены в список) или загрузить готовые файлы моделей с помощью кнопки «Импортировать+».

Для каждой сохраненной модели доступны действия:

- Скачать (↓)
- Удалить (🗑)
- Переименовать (✎)

Процессы создания и применения моделей подробно описаны в разделах Обучение модели ИИ и Использование модели ИИ.



45.3 Коннекторы

Здесь автоматически добавляются все коннекторы, созданные в результате успешного выполнения пайплайна на рабочих областях обучения и использования модели. При необходимости коннектор можно Удалить (🗑).

Подробнее о работе с ними читайте в разделе Порядок работы с коннекторами.

45.4 Дашборды

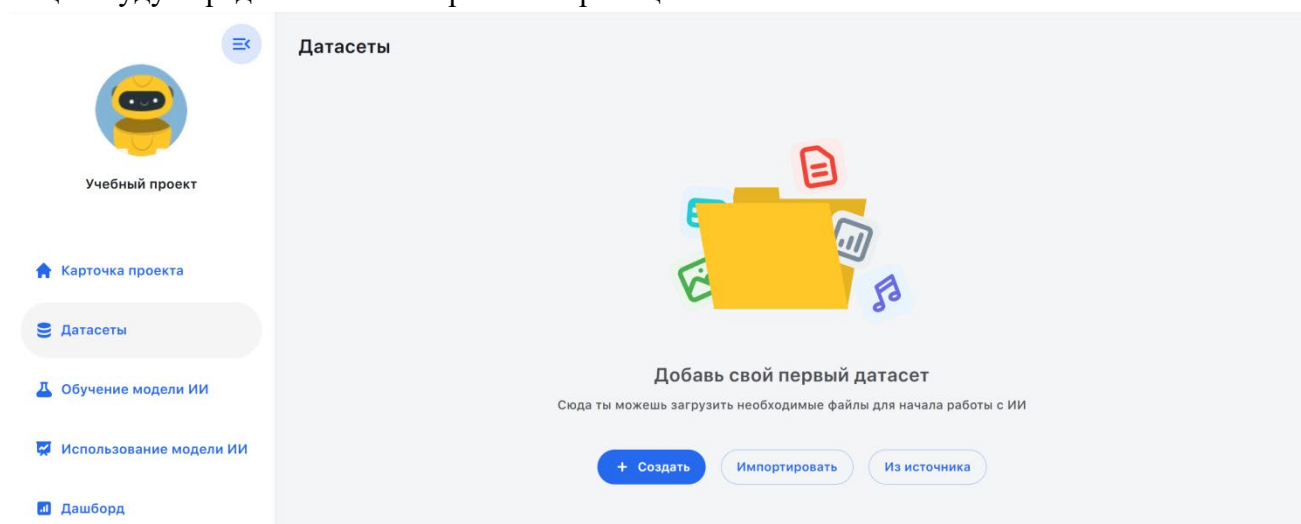
Эта вкладка содержит список всех дашбордов, созданных для визуализации данных в рамках текущего проекта. Каждый дашборд в списке кликабелен — при нажатии вы перейдете к его просмотру. Дашборд также можно Удалить (🗑).

Подробнее о создании и настройке визуализаций читайте в основном разделе Порядок работы с дашбордами.

46. Как создать датасет на основе загруженных данных

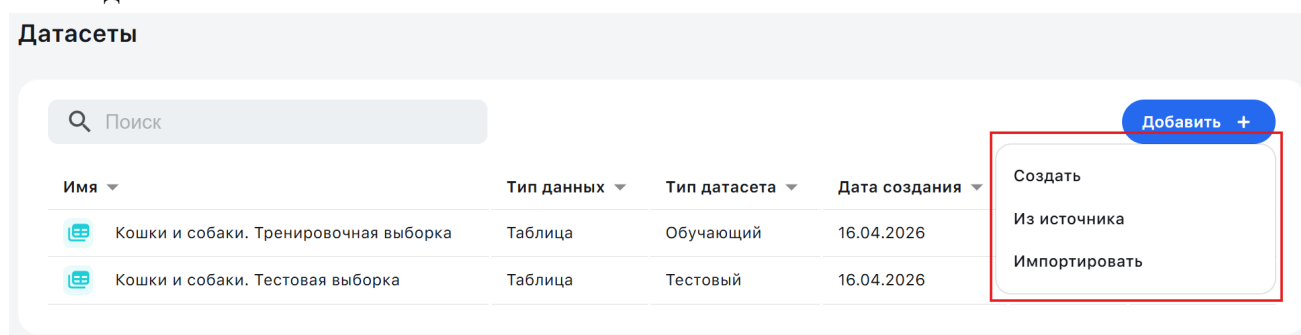
Датасет — это набор данных (например, таблицы или изображения), подготовленный для обучения или использования модели ИИ.

Чтобы использовать в проекте файлы из файлового хранилища, их необходимо сначала организовать в датасеты. Для этого перейдите в раздел «Датасеты» в боковом меню проекта. Существует три способа добавить датасет в проект. Если вы в разделе впервые, эти опции будут предложены на стартовой странице.



Если в проекте уже есть датасеты, для добавления нового нажмите кнопку «Добавить+» и выберите один из вариантов в выпадающем меню:

- Создать — формирование датасета из файлов, загруженных на платформу;
- Импортировать — быстрая загрузка готового датасета из .zip архива;
- Из источника — особый способ добавления датасета напрямую из внешней базы данных.



46.1 Создать датасет из загруженных файлов

Этот способ подходит, когда ваши данные хранятся в файловом хранилище платформы.

1. Нажмите «Создать».
2. В открывшейся форме заполните поля:
 - Название – придумайте понятное имя для вашего датасета;

- Тип данных – в выпадающем меню выберите один из типов (таблица, изображения, текст, аудио, граф, класс изображений);

Датасеты

← Назад

Новый датасет

Тип данных

Таблица

Изображения

Таблица

Текст

Аудио

Граф

Класс изображений

Тип датасета

Без типа

Добавить данные +



Выбери тип данных и загрузи файл в датасет

- Тип датасета – укажите его назначение: обучающий, тестовый или без типа (для других целей).

Датасеты

← Назад

Новый датасет

Тип данных

Таблица

Тип датасета

Без типа

Без типа

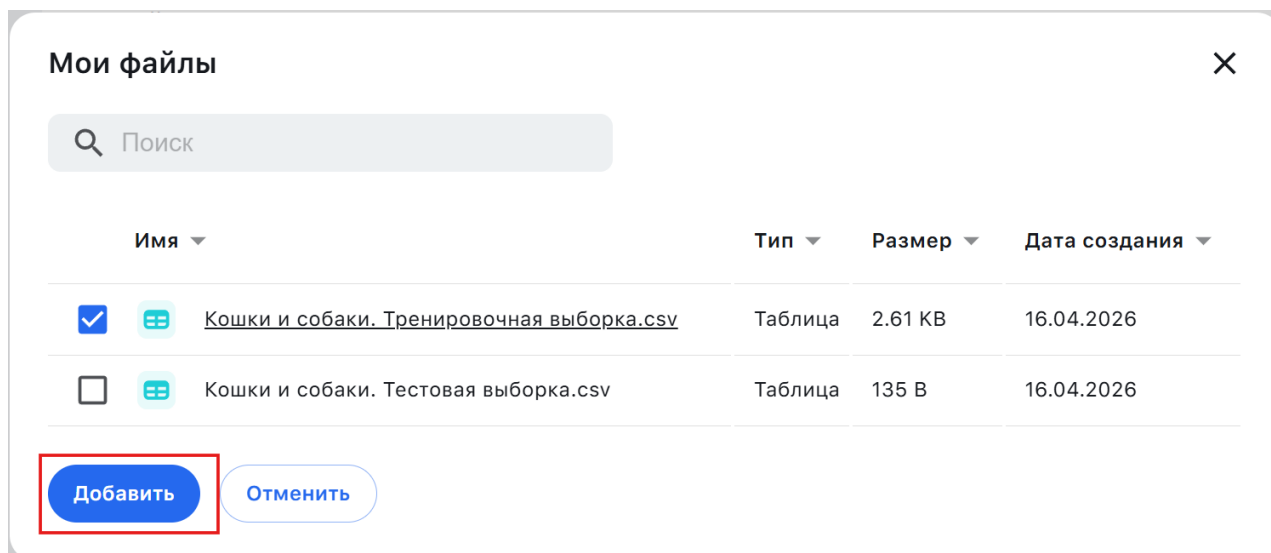
Обучающий

Тестовый



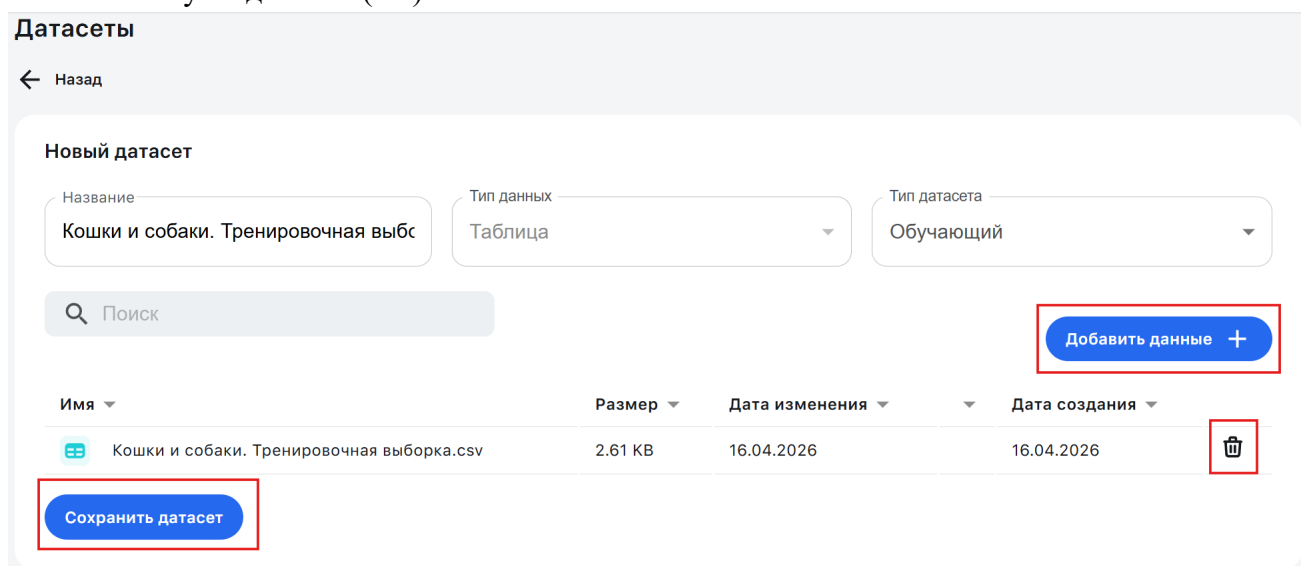
Выбери тип данных и загрузи файл в датасет

3. Нажмите на кнопку «Добавить данные+». Откроется окно вашего файлового хранилища.
4. Выберите один или несколько нужных файлов (или папок для типа «Класс изображений») и нажмите кнопку «Добавить».



5. Выбранные файлы появятся в списке. Здесь вы можете:

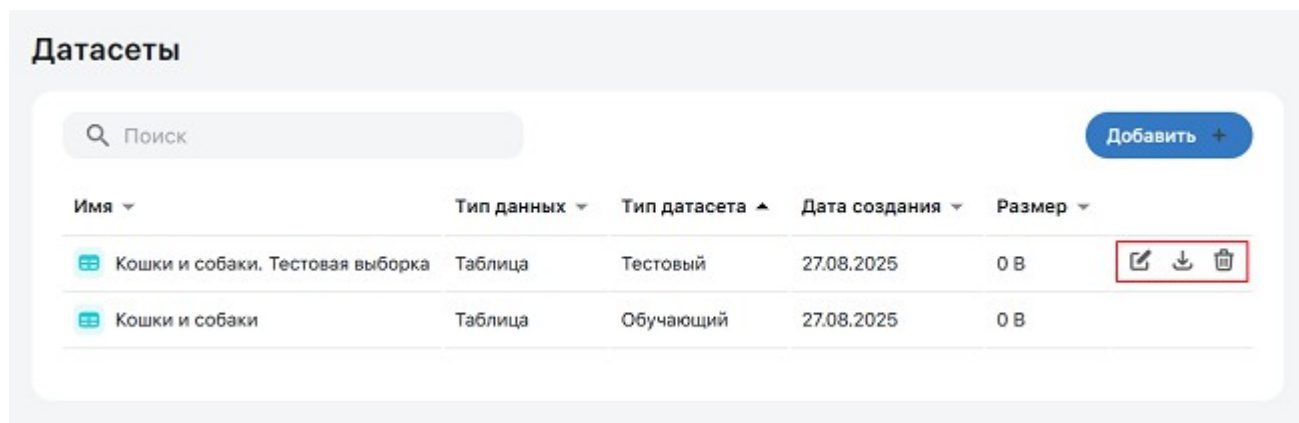
- добавить в датасет ещё файлы, снова нажав «Добавить данные+»;
- удалить ошибочно добавленные файлы или неактуальные папки, нажав на кнопку «Удалить» ().



6. Если все верно, нажмите «Сохранить датасет».

Все созданные датасеты отображаются в виде списка. При наведении курсора на строку с датасетом появятся действия для управления им:

- Редактировать (). Открывается страница настроек датасета, где можно:
 - изменить его название;
 - изменить тип датасета (например, с тестового на обучающий).
- Скачать (). Экспортирует датасет в виде .zip архива. Этот архив можно импортировать в любой другой проект на платформе.
- Удалить (). Безвозвратно удаляет датасет из проекта



46.2 Управление датасетами

Платформа Астра ИИ предоставляет гибкие инструменты для работы с датасетами, позволяя легко переносить готовые наборы данных между проектами.

Вы можете скачать датасет в виде архива и затем загрузить его в любой другой проект. Это избавит вас от необходимости вручную загружать файлы в хранилище и заново настраивать датасет.

46.2.1. Экспорт датасета

Любой созданный датасет можно выгрузить в виде .zip архива

Как экспортировать датасет:

1. Найдите нужный датасет в списке.
2. Наведите на него курсор, чтобы отобразились иконки действий.
3. Нажмите на кнопку «Скачать» ().

Скачанный .zip архив имеет стандартную структуру, которую платформа использует для корректного импорта данных. Архив содержит папку dataset и файл info_dataset.json.

1. Папка **dataset**

Содержит непосредственно файлы с данными. Ее структура зависит от типа датасета:

- Табличные данные. Внутри папки лежит один файл(.csv или .xlsx)
- Изображения. Все файлы изображений находятся внутри этой папки.
- Классы изображений. В папке dataset созданы подпапки, где имя каждой папки соответствует названию класса. Внутри каждой папки лежат соответствующие изображения.

2. Файл **info_dataset.json**

Это служебный файл с метаданными, на основе которых платформа правильно распознаёт структуру данных при импорте. Он содержит следующую ключевую информацию:

- Название датасета.
- Тип задачи (например, train или predict).
- Тип данных (table, image и т.д.).
- Пути к файлам внутри папки dataset.

Пример содержимого файла info_dataset.json:

```
{
  "dataset_data": {
    "name": "Кошки и собаки. Тренировочная выборка",
    "any_data_dict": {},
    "description": "",
    "dataset_type": "predict",
    "data_type": "table",
    "fs_paths": [
      "Кошки и собаки. Тренировочная выборка.csv"
    ],
    "dataset_category": {}
  }
}
```

46.2.2. Импорт датасета

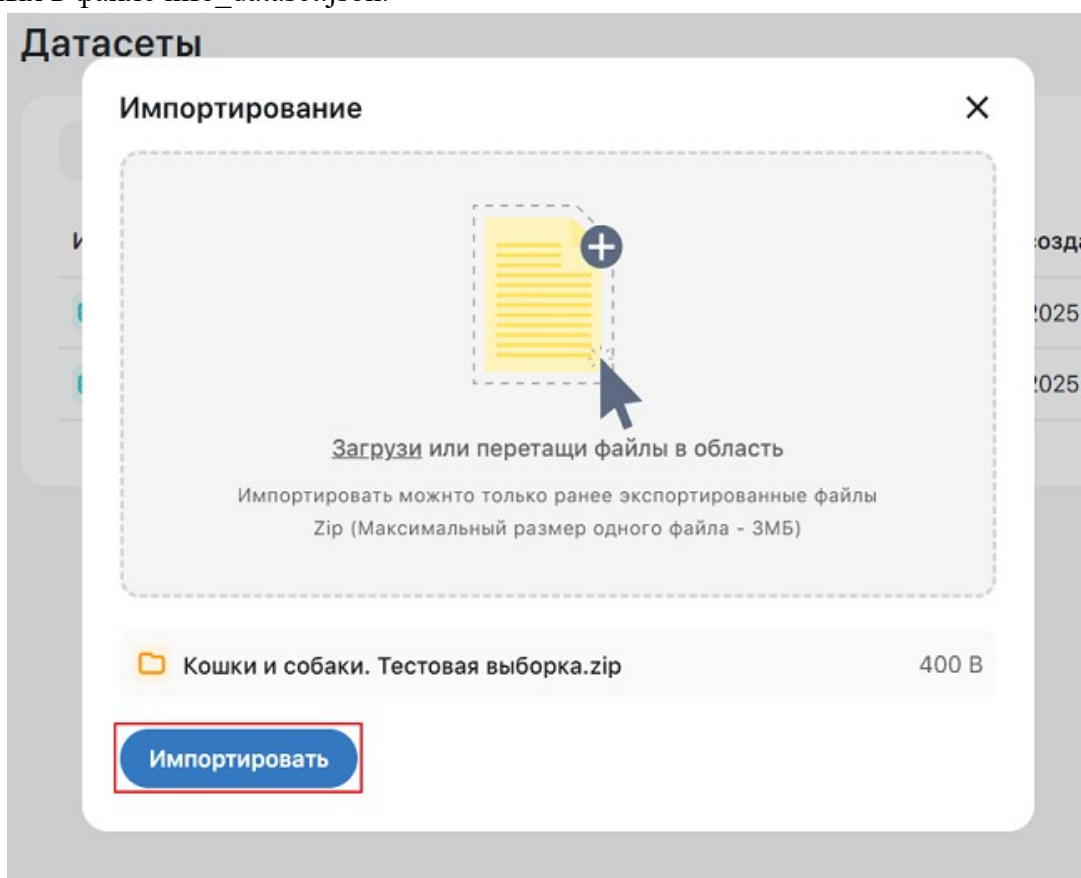
Импорт позволяет быстро добавить в проект готовый датасет, например, ранее экспортированный.

Загрузка подготовленного .zip архива происходит в обход файлового хранилища и ручной настройки, что делает этот способ удобным для переноса между проектами.

Как импортировать датасет:

1. На странице датасетов нажмите кнопку «Импортировать».
2. Перетащите .zip архив с вашего компьютера в специальную область или выберите его через файловый менеджер.
3. Нажмите «Импортировать» для завершения.

Платформа автоматически создаст новый датасет на основе данных из архива и настроек, указанных в файле info_dataset.json.



46.3 Как создать датасет из источника

Этот способ предназначен для пользователей, знакомых с SQL, и позволяет создавать датасеты, напрямую подключаясь к внешним базам данных через предварительно настроенные коннекторы. Вы можете выбрать базу данных из списка доступных подключений и сформировать датасет с помощью прямого SQL-запроса.

Шаг 1: Выбор базы данных

Выберите вкладку «Датасеты» в боковом меню, нажмите «Добавить +» и выберите «Из источника». Откроется рабочая область для создания запроса.

Общие настройки

Название датасета

Тип датасета
Без типа

☒ Обновляемый

Интервал *
60

База данных

Источник данных

Схема

SQL-запрос



Здесь пока пусто

Выбери источник данных, чтобы посмотреть схему

В выпадающем списке «База данных» выберите необходимое подключение из списка доступных для вашего проекта.

Обновляемый

Интервал *

60

База данных

Источник данных

Фильмы

Схема

SQL-запрос



После выбора базы данных вы можете ознакомиться с её структурой на вкладке «Схема». Здесь наглядно представлены все таблицы, их поля, типы данных и связи между ними. Это поможет вам правильно составить SQL-запрос, используя верные названия таблиц и столбцов.

Шаг 2: Формирование SQL-запроса

Перейдите на вкладку «SQL-запрос» и в текстовом поле введите ваш запрос. Редактор подсвечивает синтаксис SQL, что помогает избежать ошибок при написании.

Вы можете работать с несколькими запросами одновременно. Нажмите на «+», чтобы добавить новую вкладку для другого запроса (SQL2, SQL3 и т.д.). Это удобно для тестирования разных вариантов выборки данных.

База данных

База данных
Фильмы

Схема

SQL-запрос

SQL1

+

SELECT * FROM фильмы;

Выполнить запрос

Сохранить изменения

Шаг 3: Выполнение запроса и предварительный просмотр

Чтобы проверить результат, нажмите «Выполнить запрос». Ниже появится область «Предварительного просмотра» с таблицей, содержащей первые строки полученных данных.

Убедитесь, что выборка составлена верно. Если она не соответствует ожиданиям, вы можете изменить SQL-запрос и запустить его повторно.

Чтобы сохранить текст вашего SQL-запроса для дальнейшего использования, нажмите «Сохранить изменения». Вы сможете вернуться к нему позже для редактирования или повторного выполнения.

SELECT * FROM фильмы;

Выполнить запрос

Сохранить изменения

Предварительный просмотр

id_фильма	id_языка	название	рейтинг	год_выпуска	популярность	стоимость_проката
1	1	ACADEMY DINOSAUR	PG	2012		0.99
2	1	ACE GOLDFINGER	G	2023		4.99
3	2	ADAPTATION HOLES	NC-17	2017		2.99
4	6	AFFAIR PREJUDICE	G	2023		2.99
5	4	AFRICAN EGG	G	2019		2.99
6	1	AGENT TRUMAN	PG	2010		2.99
7	1	AIRPLANE SIERRA	PG-13	2019		4.99
8	1	AIRPORT POLLOCK	R	2010		4.99
9	1	ALABAMA DEVIL	PG-13	2011		2.99
10	1	ALADDIN CALENDAR	NC-17	2014		4.99

Сохранить результат

Шаг 4: Сохранение результата в датасет

Если данные в таблице предварительного просмотра вас устраивают, нажмите «Сохранить результат».

Шаг 5: Проверка

После сохранения вернитесь на главную страницу раздела «Датасеты». Ваш новый датасет, созданный из источника, появится в общем списке и будет готов к использованию в пайплайнах.

47. Обучение модели ИИ

После того, как данные подготовлены, можно приступить к построению пайплайнов для обучения и использования моделей ИИ. Вся работа происходит на специальных рабочих областях.

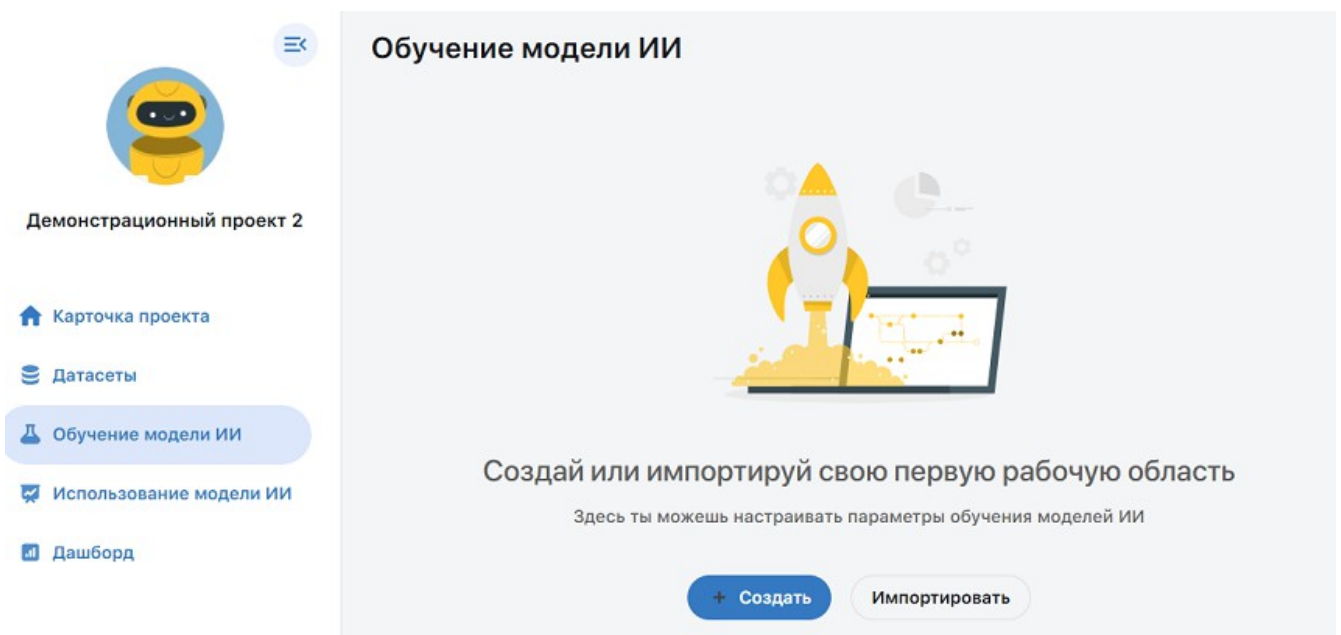
Рабочая область – это пространство для построения пайплайнов (pipelines). Пайплайн представляет собой последовательность блоков, соединенных в единую блок-схему.

Чтобы приступить к работе, перейдите в меню слева и выберите пункт «Обучение модели ИИ».

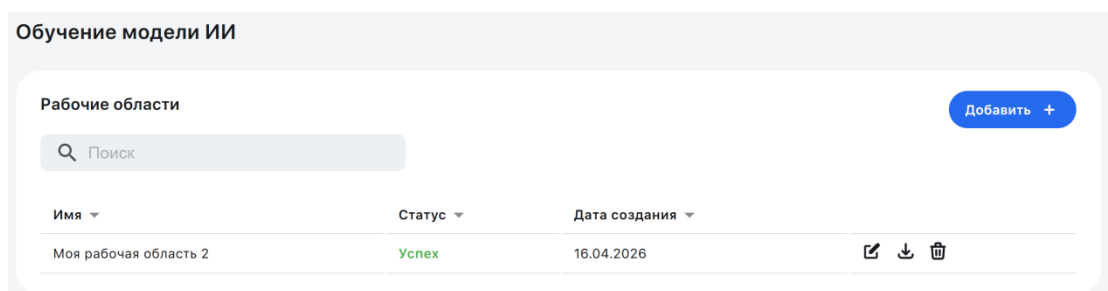
47.1 Начало работы: создаем рабочую область

Если в проекте еще нет ни одной рабочей области, вы увидите стартовый экран. Вы можете:

- Создать рабочую область : построить свой пайплайн с нуля.
- Импортировать рабочую область: загрузить ранее сохраненную рабочую область (как это сделать, мы рассмотрим ниже).



Чтобы создать новую рабочую область, нажмите «+ Создать», введите её название и подтвердите действие. После этого откроется пустая рабочая область. Если у вас уже есть рабочие области, они будут отображены в виде списка.



47.2 Управление рабочими областями

При наведении курсора на строку рабочей области станут доступны опции для управления:

- Редактировать (✎): изменить название рабочей области.
- Скачать (↓): экспортировать рабочую область в виде zip архива. Подробнее.
- Удалить (🗑): безвозвратно удалить рабочую область и все её содержимое.

Вы можете легко сохранять рабочие области на свой компьютер и загружать их обратно на платформу. Это очень удобно для создания резервных копий важных пайплайнов, переноса рабочих областей между разными проектами или обмена готовыми пайплайнами с коллегами.

47.2.1. Экспорт рабочей области

1. Наведите курсор на нужную рабочую область в списке.
2. Нажмите «Скачать».
3. Браузер загрузит рабочую область в zip архив. Имя архива по умолчанию формируется из названия вашей рабочей области. При необходимости вы можете указать любое другое название файла перед сохранением.

Экспортированный .zip архив — это самодостаточный снимок вашего пайплайна. Он содержит Файлы конфигурации, которые в совокупности описывают все его компоненты: структуру пайплайна и настройки блоков, и копии данных, использованных в блоках.

Содержимое архива:

1. Папка **datasets**. Содержит фактические файлы данных (например, .xlsx, .csv), которые используются в пайплайне. Каждый датасет помещается в отдельную подпапку с уникальным именем (UUID). Это гарантирует, что при импорте рабочая область будет иметь все необходимые данные и не будет зависеть от их наличия в целевом проекте.
2. **workspaces.json**. Главный конфигурационный файл. Он описывает саму рабочую область и определяет структуру пайплайна.
3. **blocks.json**. Содержит детальную конфигурацию для каждого блока, чей UUID упомянут в workspaces.json.
4. **datasets.json**. Этот файл отвечает за метаданные, связанные с датасетами внутри пайплайна.
5. **models.json**. Файл отвечает за метаданные, связанные с моделями внутри пайплайна.

47.2.2. Импорт рабочей области

1. На странице со списком рабочих областей нажмите «Добавить+» и выберите Импортировать (или кнопку Импортировать на стартовом экране, если у вас еще нет рабочих областей).
2. В открывшемся окне выберите zip архив, который вы ранее экспортировали.
3. После успешной загрузки вы увидите уведомление, а импортированная рабочая область появится в общем списке.

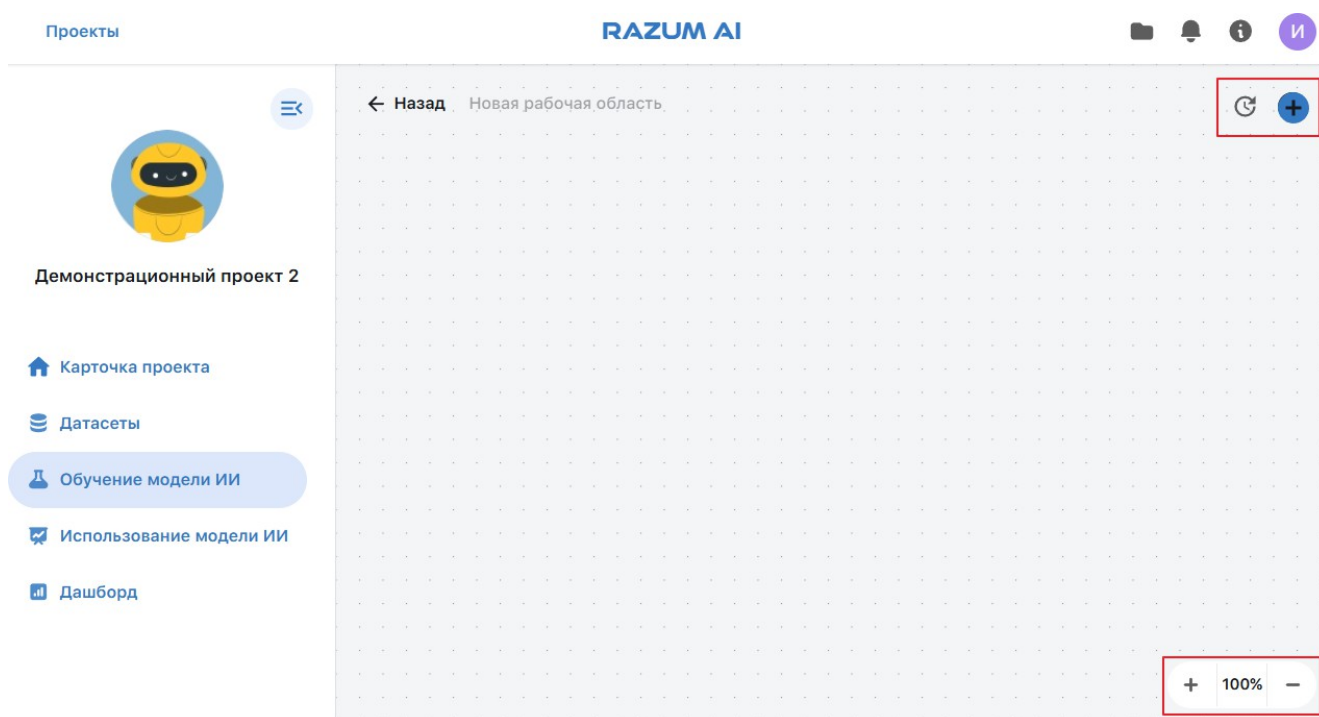
Теперь вы можете открыть её и продолжить работу. Весь пайплайн будет полностью восстановлен в том виде, в котором он был на момент экспорта. Все связанные датасеты из

архива будут автоматически добавлены в раздел Датасеты вашего проекта, что обеспечивает полную работоспособность пайплайна «из коробки».

47.3 Обзор рабочей области

Рабочая область состоит из нескольких ключевых элементов:

- Холст: основное поле, где вы размещаете и соединяете блоки.
 - Панель управления: кнопки внизу для управления рабочей областью.
 - Панель действий:
- Добавить блок (+): открывает панель добавления блоков: меню справа, содержащее все доступные блоки.
 - История запусков (↺): открывает историю запусков пайплайна. Из успешных можно создать коннекторы для визуализации данных.



47.3.1. Панель добавления блоков

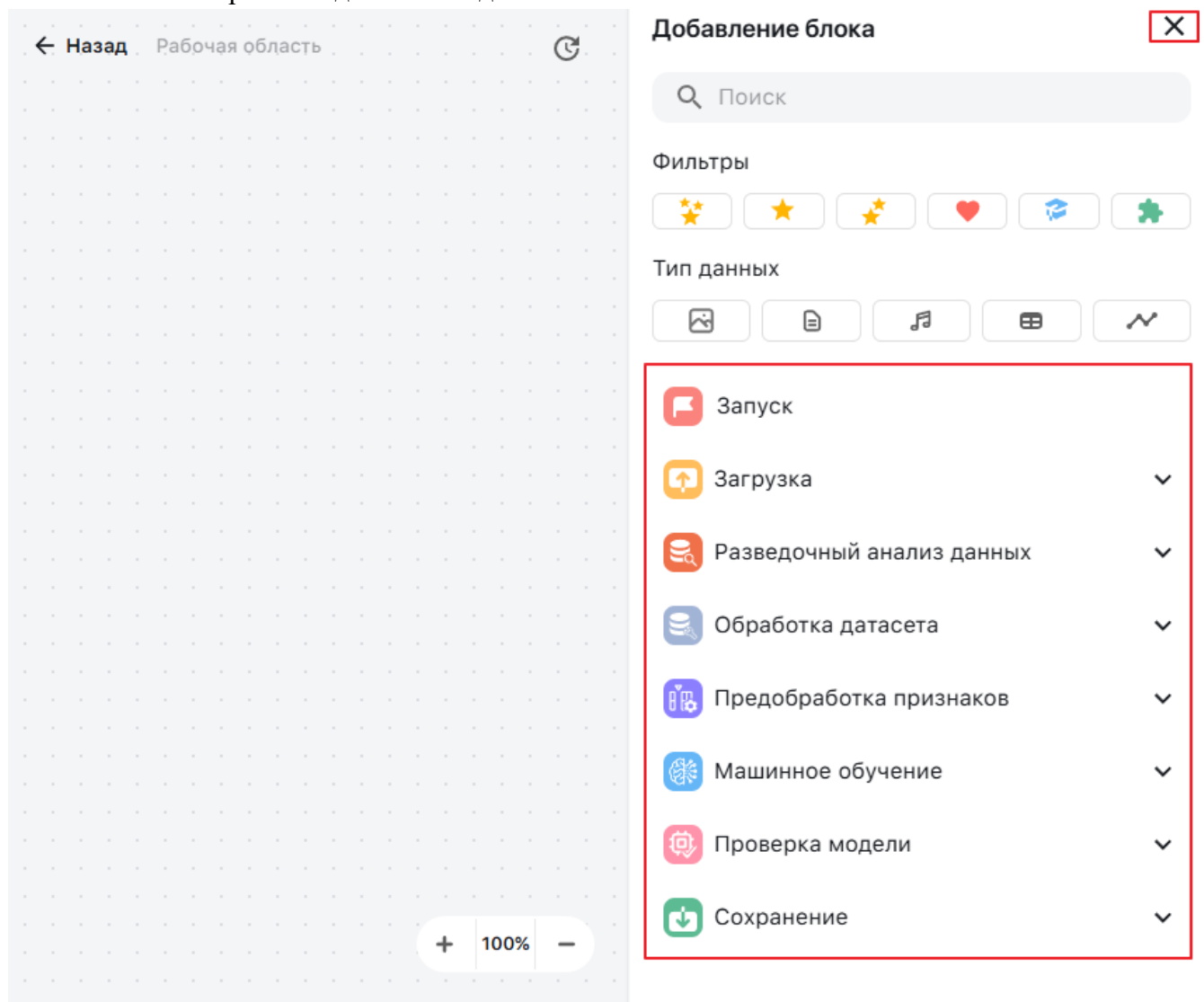
Чтобы открыть список доступных блоков на рабочей области, необходимо нажать на кнопку, расположенную в правом верхнем углу.

Для того, чтобы расположить блок на рабочей области его необходимо перетащить из меню добавления блоков, зажав левую клавишу мыши (ЛКМ). Чтобы закрыть панель с блоками – нажмите крестик в правом углу.

Все блоки для удобного использования разделены на тематические группы и выделены цветом.

- Загрузка данных
- Разведочный анализ данных (EDA)
- Обработка датасета (общая подготовка данных)
- Предобработка признаков (подготовка столбцов (признаков))
- Машинное обучение (алгоритмы МО)

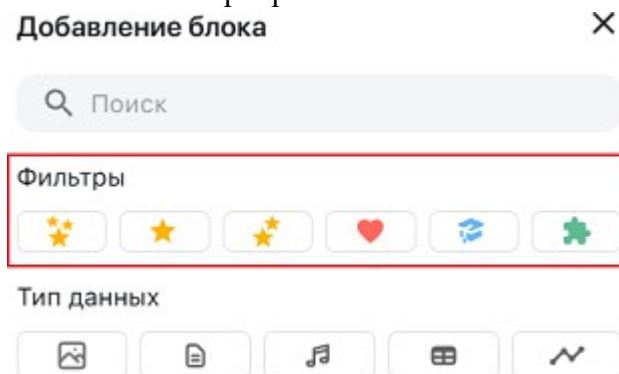
- Проверка модели (валидация)
- Сохранение датасета/модели



47.3.2 Фильтры

Панель содержит удобные инструменты для поиска нужных блоков. При наведении курсора на любую иконку в панели фильтров появляется всплывающая подсказка с её названием.

Категория Фильтры. Позволяет отсортировать блоки по их сложности и назначению.

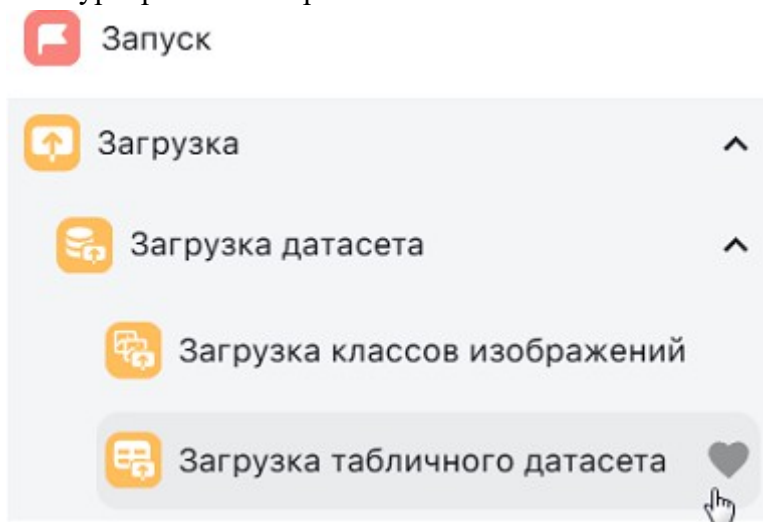


	Легкий	Фильтр для отображения базовых блоков, которые подходят для начинающих или для выполнения простых, стандартных операций.
	Средний	Отображает блоки средней сложности, требующие базового понимания процессов анализа данных и машинного обучения.
	Сложный	Фильтр для продвинутых пользователей. Включает сложные алгоритмы и инструменты, требующие глубоких знаний.
	Онлайн-кейсы	Показывает блоки, которые используются в готовых практических проектах -онлайн-кейсах.
	Избранное	Ваш личный фильтр. Позволяет отобразить только те блоки, которые вы отметили как избранные.

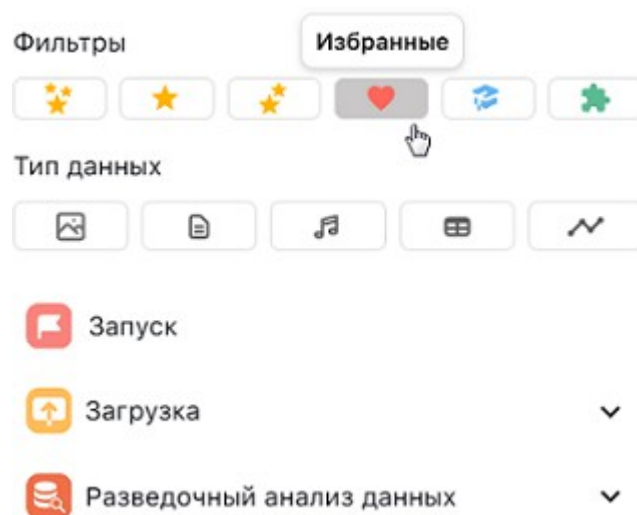
Как работает фильтр Избранное

Добавить в Избранное — это удобный способ создать личную подборку часто используемых блоков для быстрого доступа.

1. Найдите нужный блок в общем списке.
2. Наведите на него курсор мыши. Справа от названия блока появится серое сердечко:

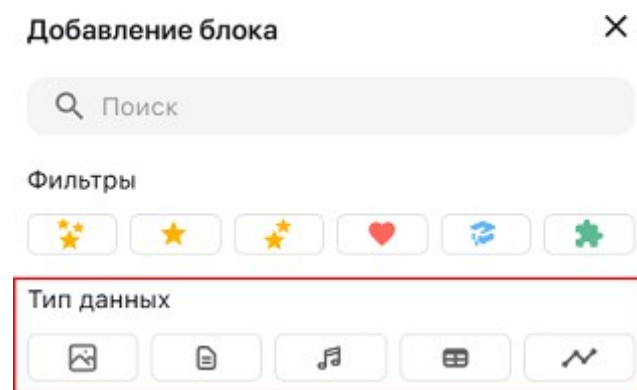


3. Нажмите на это сердечко. Оно станет красным, что означает, что блок успешно добавлен в ваш список избранных.
4. В панели фильтров выберите «Избранное».
5. Останутся только те блоки, которые вы ранее отметили:



Категория Тип данных

Эти фильтры помогут найти блоки, которые предназначены для работы с определённым типом данных.



	Изображения	Фильтрует блоки, предназначенные для обработки изображений.
	Текст	Фильтрует блоки для работы с текстовыми данными.
	Таблица	Фильтр для продвинутых пользователей. Включает сложные алгоритмы и инструменты, требующие глубоких знаний.

Чтобы отфильтровать блоки по типам данных, нажмите на одну или несколько иконок. В результате отобразятся только подходящие блоки.

47.4 Процесс сборки пайплайна

47.4.1. Добавление и перемещение блоков

Чтобы добавить блок на рабочую область, найдите его в боковом меню и перетащите, зажав левую кнопку мыши (ЛКМ).

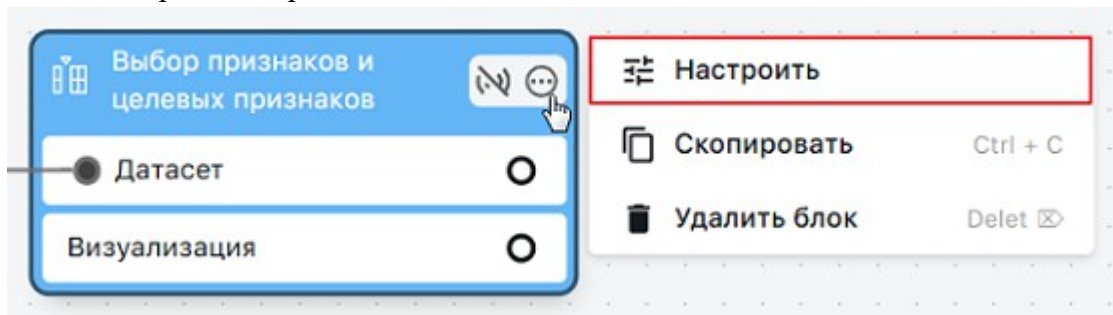
- Чтобы переместить блок, наведите на него курсор, зажмите ЛКМ и перетащите в нужное место.

- Чтобы сдвинуть всю рабочую области, нажмите ПКМ на любом свободном месте и двигайте курсор в необходимом направлении.

47.4.2. Настройки блока

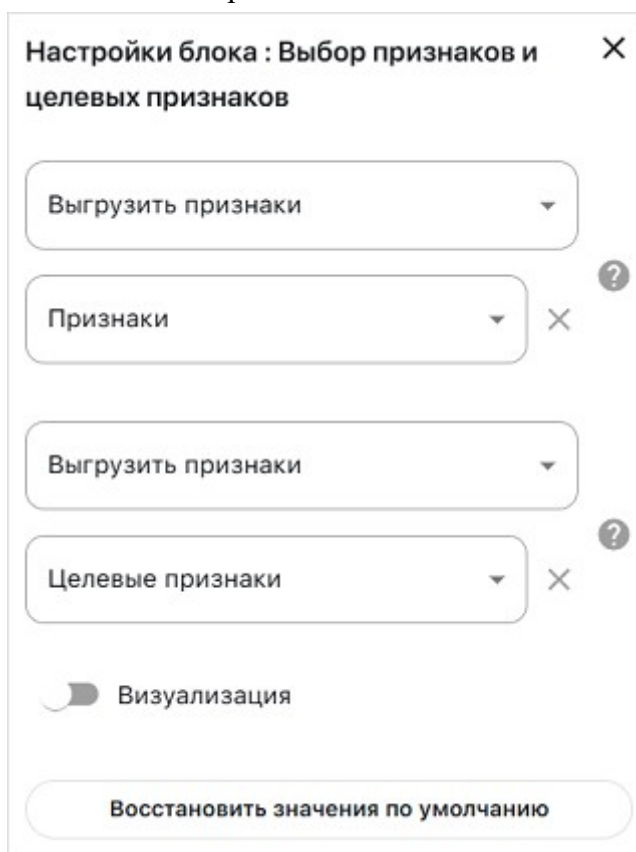
Каждый блок имеет индивидуальные параметры. Открыть меню настроек вы можете двумя способами:

- нажмите на кнопку с троеточием в правом верхнем углу блока, в выпадающем меню выберите настроить;

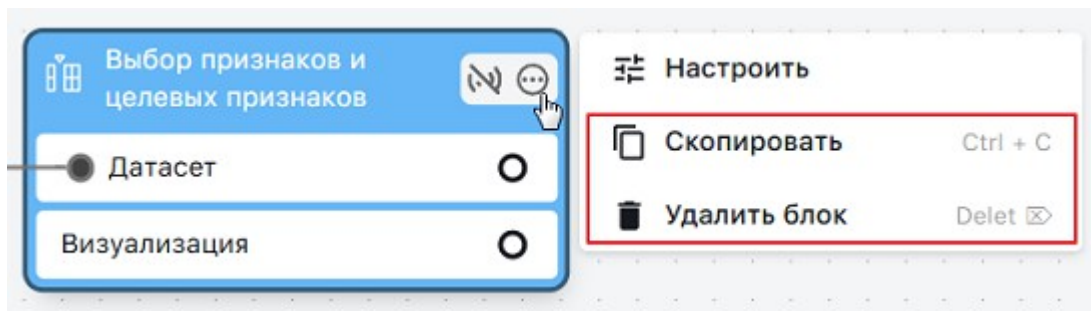


- два раза кликните по блоку.

В результате откроется меню с настройками:



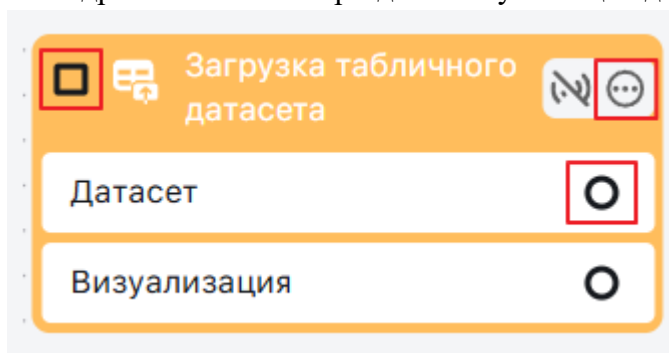
Чтобы удалить или скопировать блок, также нажмите на кнопку с троеточием в правом углу блока и в выпадающем меню выберите нужный пункт.



47.4.3. Соединение блоков

Каждый блок имеет порты для связи с другими блоками:

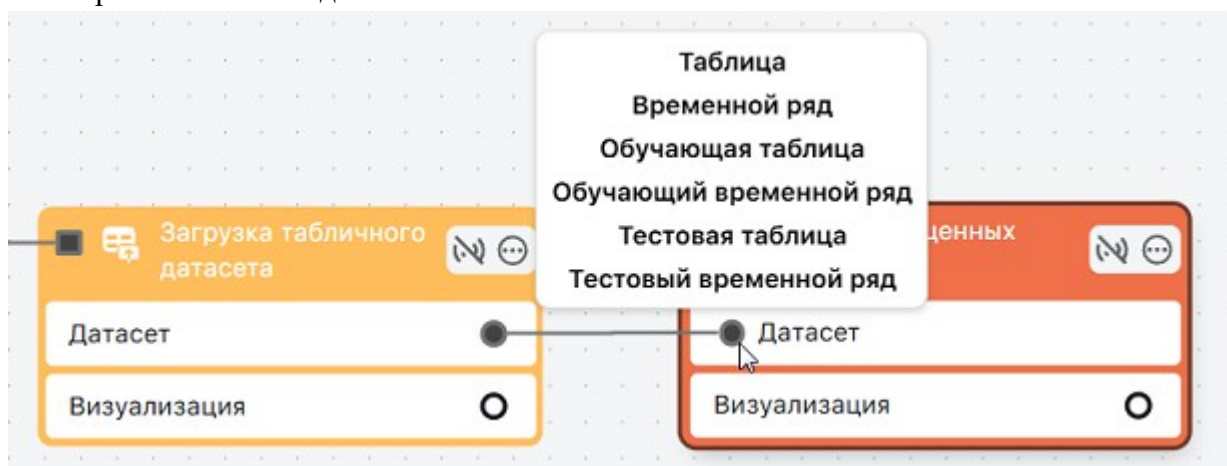
- Входной порт (■/●): принимает данные от предыдущего блока.
- Выходной порт (●): передаёт обработанные данные следующему блоку.
- Выход для визуализации (○): специальный порт, который не соединяется с другими блоками. Подробнее об этом в разделе Визуализация данных.



Чтобы соединить блоки:

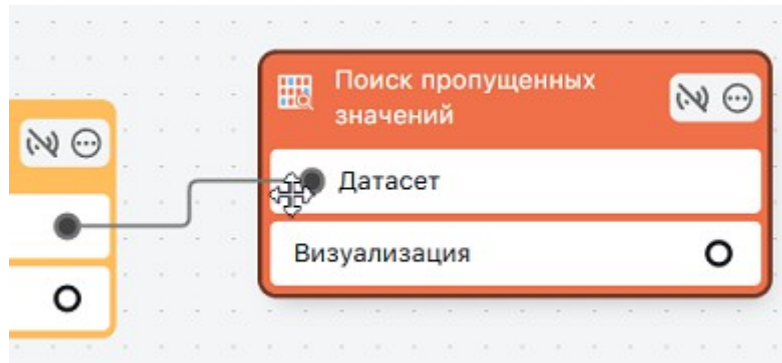
1. Нажмите ЛКМ на выходной порт (●) одного блока.
2. Протяните линию ко входному порту (■/●) следующего блока и отпустите кнопку.

Каждый блок принимает на вход и выдает на выход данные разных семантических типов. При наведении курсора на каждый порт появляется подсказка, какие типы данных отдает блок, а какие принимает на вход.



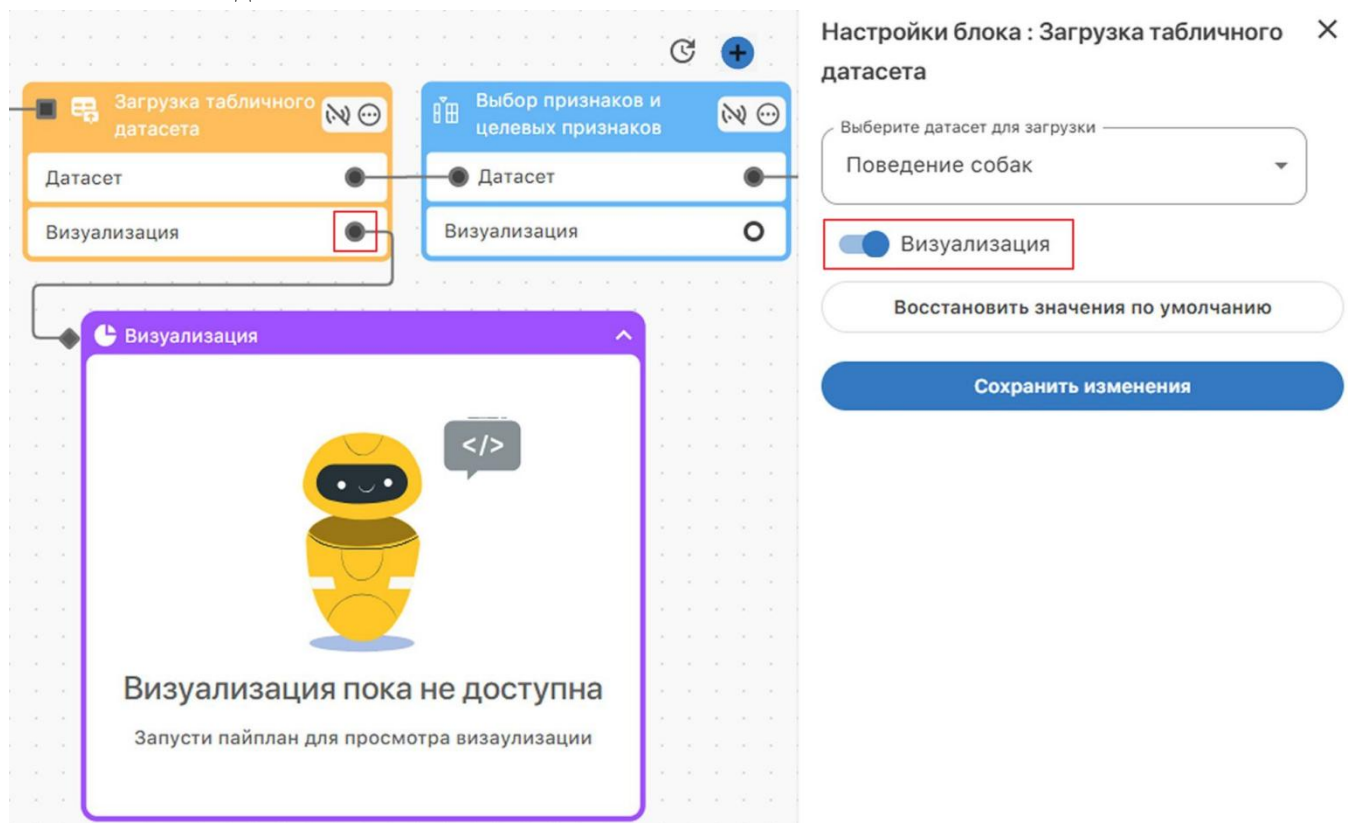
47.4.4. Удаление связей

Чтобы удалить связь между блоками, наведите курсор на линию рядом с одним из портов, чтобы появился значок «+», нажмите на неё ЛКМ. Связь исчезнет.



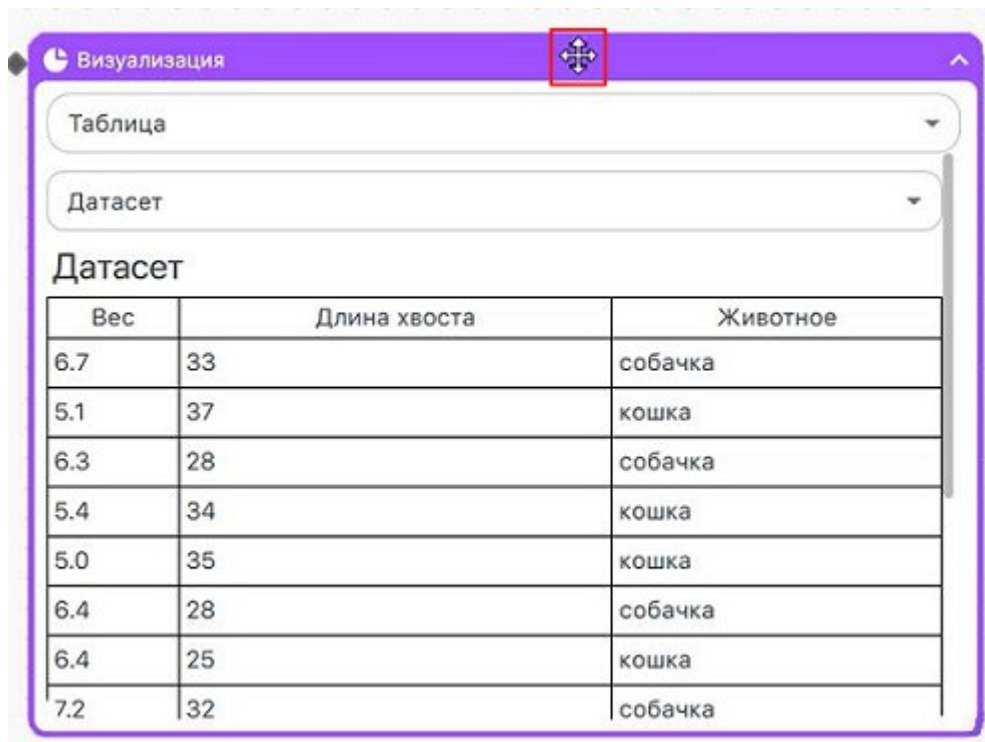
47.4.5. Визуализация

У всех блоков в панели настроек есть переключатель «Визуализация». Как только вы его включите, у блока активируется выходной порт визуализации и на рабочей области появляется соответствующее окно. Блок Визуализация предназначен для просмотра результатов выполнения каждого блока.



Изначально в окне будет отображаться сообщение «Визуализация пока не доступна». После успешного запуска пайплайна в нём появятся данные.

Чтобы изменить положение окна визуализации наведите на него курсор мыши, захватите и можете передвигать.



Вес	Длина хвоста	Животное
6.7	33	собачка
5.1	37	кошка
6.3	28	собачка
5.4	34	кошка
5.0	35	кошка
6.4	28	собачка
6.4	25	кошка
7.2	32	собачка

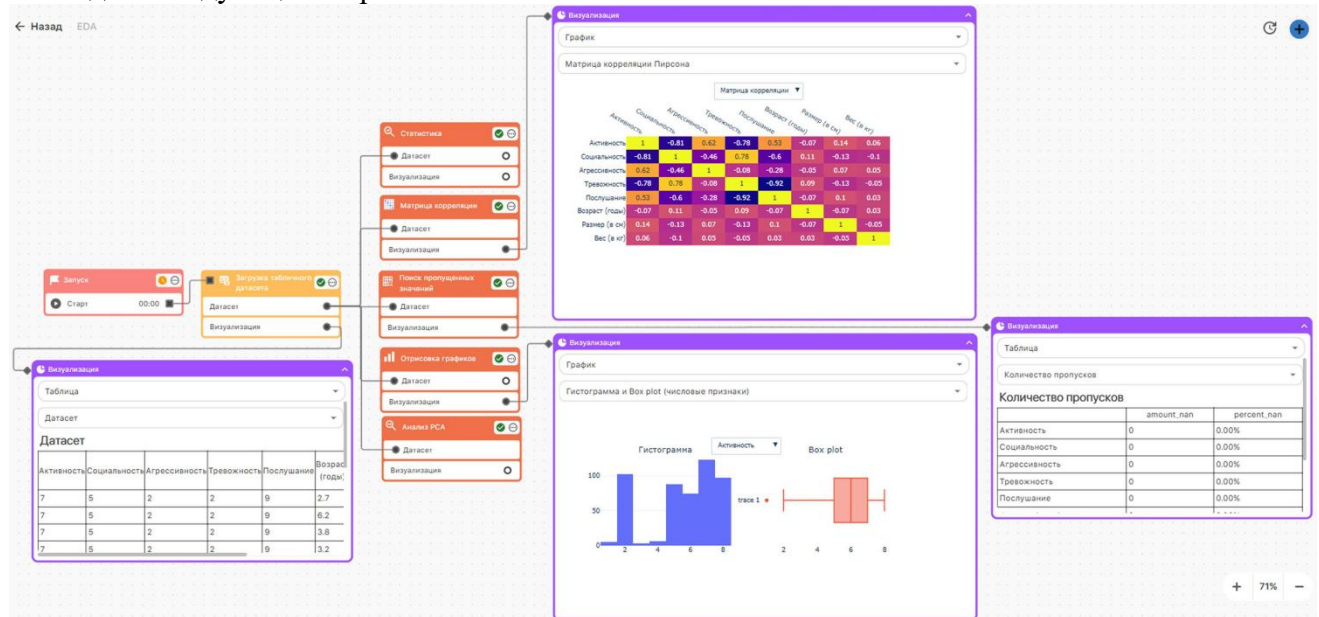
Изменение размера окна

Окно блока Визуализация является интерактивным, и его размер можно изменять для удобства просмотра. Наведите курсор на край окна визуализации, нажмите левую кнопку мыши и потяните, чтобы изменить размер.

48. Практические примеры. Разведочный анализ данных

Прежде чем обучать модель, важно изучить данные, с которыми вы работаете. Блоки из раздела «Разведочный анализ данных» позволяют исследовать датасет: найти пропуски, посмотреть на распределение признаков и оценить корреляцию между ними. Этот этап помогает принять обоснованные решения о том, какие данные требуют предобработки и какие признаки будут наиболее полезны для модели.

Для проведения разведочного анализа последовательно соберем пайплайн, который будет выглядеть следующим образом:

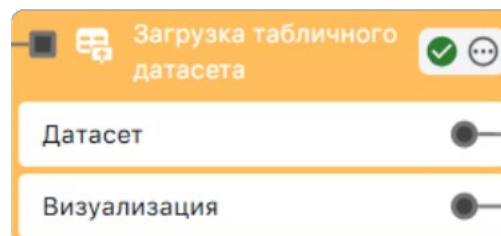


48.1 Загрузка данных

Первый шаг любого проекта — загрузка данных. Загрузим заранее созданный табличный датасет с помощью блока «Загрузка табличного датасета».

Настройка

Нажмите на иконку настроек и в выпадающем списке «Выберите датасет для загрузки» укажите нужный файл, в нашем случае — «Поведение собак».



Настройки блока : Загрузка табличного датасета

Выберите датасет для загрузки

Поведение собак



Визуализация

Ниже представлена визуализация загруженных данных. Таблица содержит следующие столбцы:

- Поведенческие признаки: Активность, Социальность, Агрессивность, Тревожность, Послушание.
- Физические параметры: Возраст (годы), Размер (в см), Вес (в кг).
- Общая информация: Порода (группа), Пол.

Визуализация									
Таблица									
Датасет									
Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)	Порода (группа)	Пол
7	5	2	2	9	2.7	59.5	22.8	служебная	Самец
5	5	2	2	9	6.2	70.6	19.1	служебная	Самец
7	5	2	2	9	3.8	18.3	27.0	служебная	Самец
7	5	2	2	9	5.3	56.4	16.7	служебная	Самка
7	5	2	2	9	6.1	63.1	20.4	служебная	Самец
7	5	2	2	9	6.3	68.9	10.9	служебная	Самка
7	5	2	2	9	8.5	35.6	11.0	служебная	Самка
7	5	2	2	9	1.7	83.1	19.2	служебная	Самец
7	5	2	2	9	4.1	33.1	20.3	служебная	Самец
7	5	2	2	9	0.5	29.8	1.8	служебная	Самец
7	5	2	2	9	8.7	44.0	21.5	служебная	Самец
7	5	2	2	9	7.5	22.5	12.9	служебная	Самка
7	5	2	2	9	9.2	52.1	2.8	служебная	Самка
7	5	2	2	8	4.5	41.5	20.6	служебная	Самец

48.2 Анализ корреляций

Используем блок «Матрица корреляции», чтобы понять, как различные характеристики собак связаны друг с другом.

Настройка

Режим: «Упрощённый режим» выключим для ручного контроля и настройки набора анализируемых признаков.

Тип корреляции: Поскольку основная цель разведочного анализа — изучить взаимосвязи между числовыми оценками поведения и физическими параметрами, выбираем «Корреляцию Пирсона».

Признаки: В поле «Признаки» выбираем все интересующие числовые столбцы.

Матрица корреляции

Датасет

Визуализация

Настройки блока : Матрица корреляции

☐ Упрощённый режим

Тип корреляции
Корреляция Пирсона

Выгрузить признаки
Поведение собак

Признаки

Активность × Социальность ×

Агрессивность ×

Тревожность × Послушание ×

Возраст (годы) ×

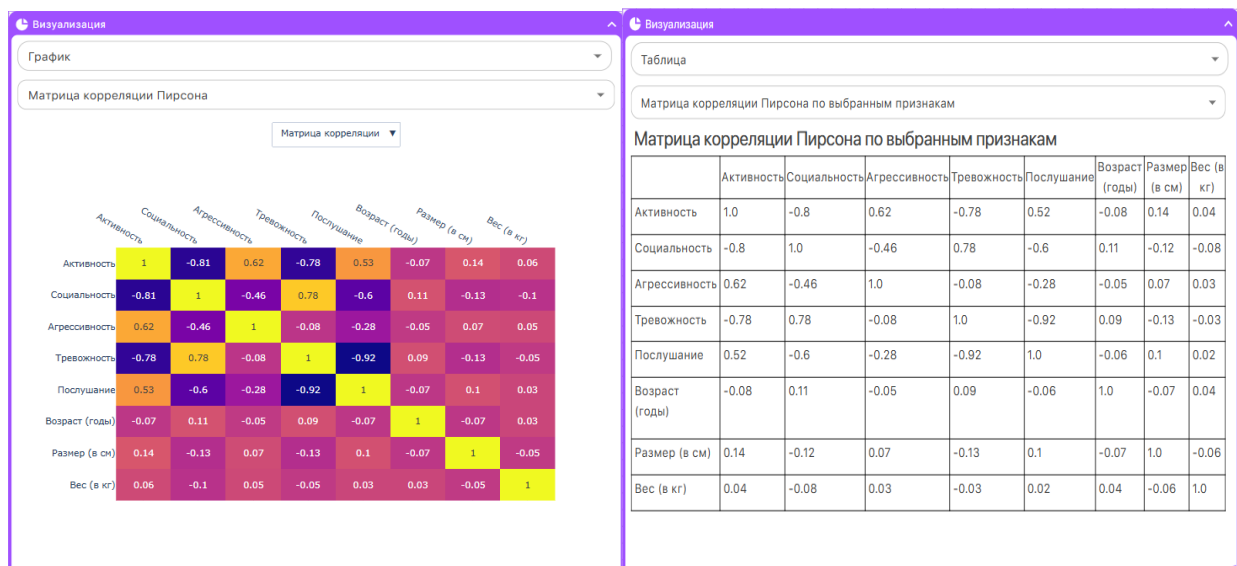
Размер (в см) × Вес (в кг) ×

☒ Визуализация

48.2.1 Анализ результатов

Визуализация выводит тепловую карту и таблицу с коэффициентами корреляции Пирсона.

- Сильная отрицательная связь наблюдается между признаками Послушание и Тревожность (-0.92).
- Выраженная положительная связь — между Тревожностью и Социальностью (0.78).
- Также заметна сильная обратная связь между Активностью и Социальностью (-0.81).
- Физические параметры (Возраст, Размер, Вес) очень слабо коррелируют с поведенческими характеристиками.



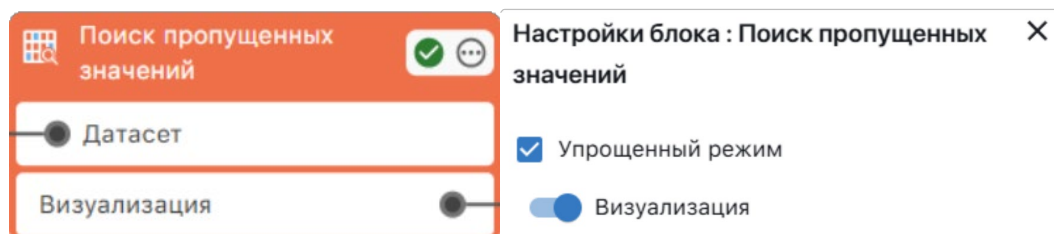
Вывод: анализ показывает, что поведенческие признаки сильно связаны между собой, в то время как физические параметры (возраст, вес, размер) практически не коррелируют с поведением в данной выборке. Например, уровень тревожности сильнее всего связан с другими поведенческими чертами: высокая тревожность соответствует низкому послушанию и, наоборот, высокой социальности.

48.3 Поиск пропущенных значений

Чистота данных — залог успешной модели. Блок «Поиск пропущенных значений» сканирует датасет на наличие пустых ячеек, которые могут помешать анализу.

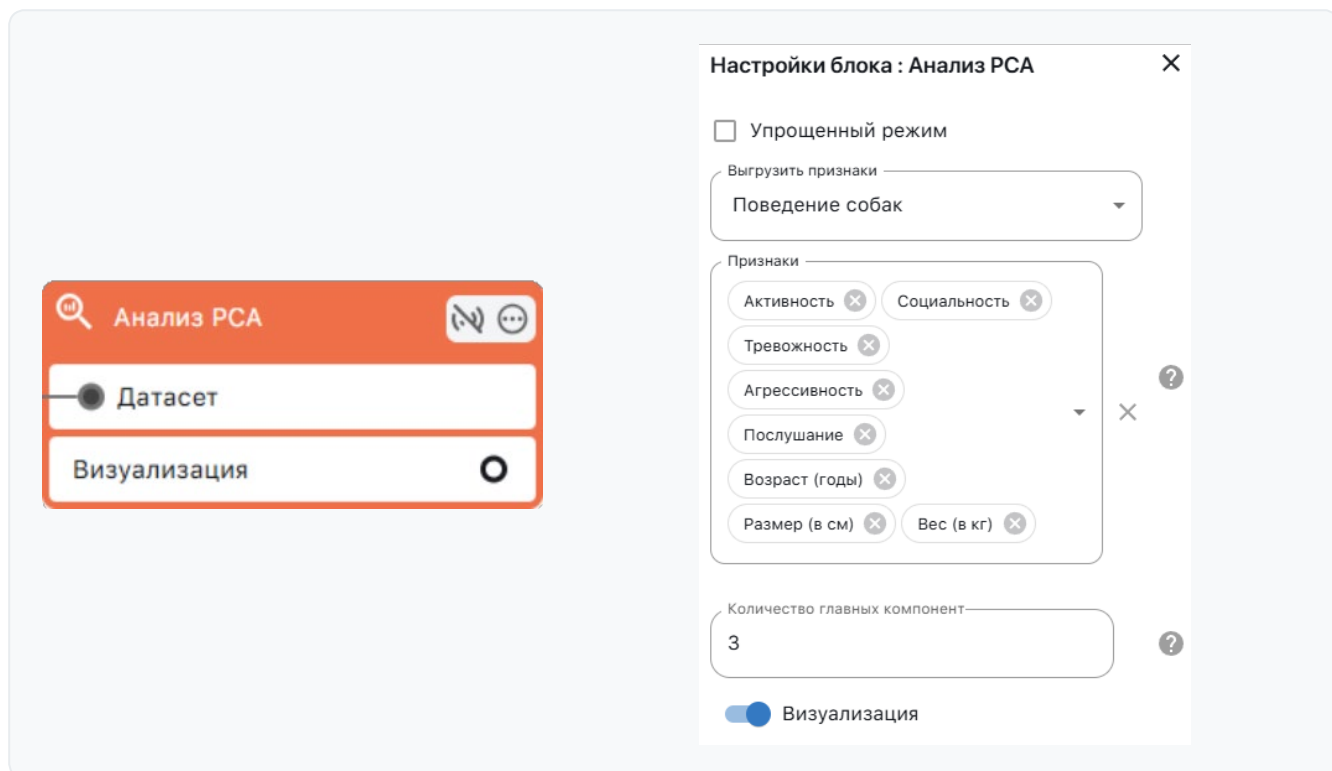
Настройка

Используем «Упрощенный режим», который автоматически проверяет все столбцы в датасете, избавляя от необходимости выбирать их вручную.



48.3.1. Анализ результатов

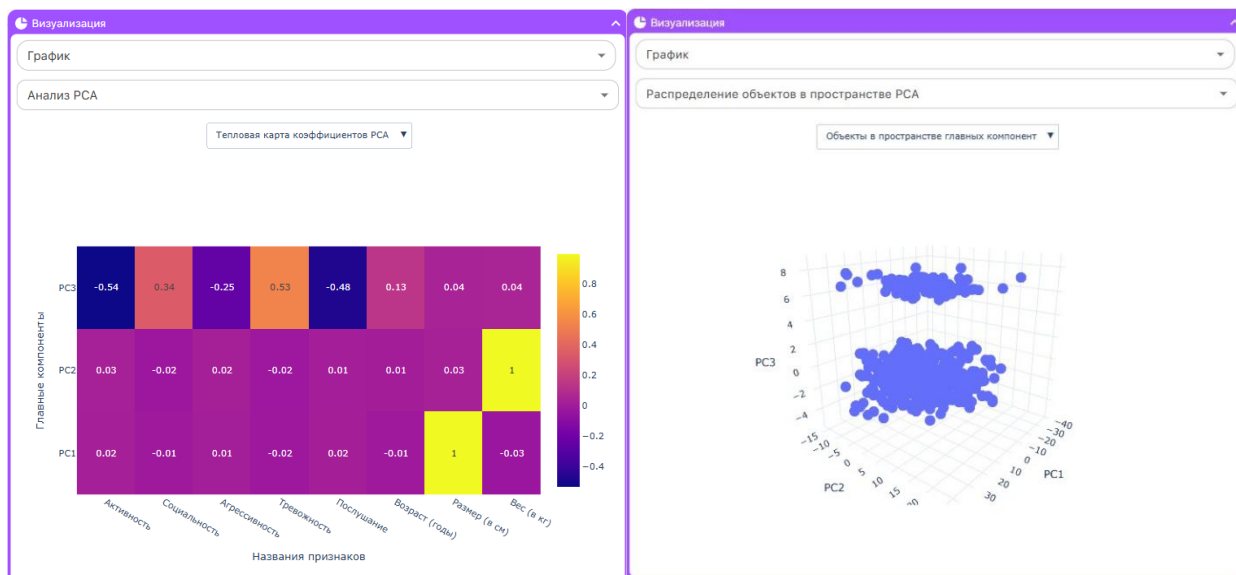
Таблица с результатами показывает количество (amount_nan) и процент (percent_nan) пропущенных значений для каждого столбца. Как видно, ни в одном из признаков нет пропущенных значений (NaN).



48.4.1. Анализ результатов

Визуализация метода главных компонент (PCA) показывает следующее:

1. По тепловой карте коэффициентов:
 - Первая компонента (PC1) определяется признаком Возраст (годы).
 - Вторая компонента (PC2) соответствует признаку Вес (в кг).
 - Третья компонента (PC3) является смешанной поведенческой характеристикой, на которую наибольшее влияние оказывают Тревожность (0.53), Активность (-0.54) и Послушание (-0.48).
2. По 3D-графику распределения:
 - Четко видны два отдельных кластера (группы).
 - Эти группы разделены в основном по вертикальной оси (PC3). Это ключевой момент: он означает, что именно поведенческие характеристики разделяют всех собак на две большие, разные группы.



Вывод: PCA выявил, что в датасете существуют две четко выраженные группы собак.

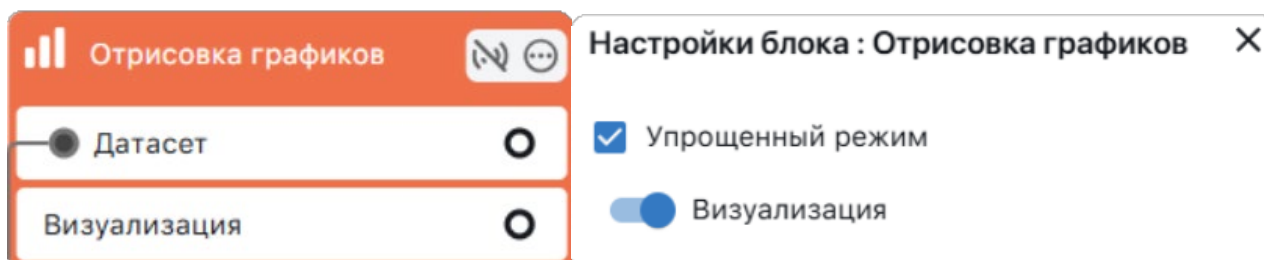
48.5 Визуальный анализ распределений

Блок «Отрисовка графиков» — наш главный инструмент для детального изучения каждого признака.

Настройка

Используем «Упрощенный режим». Блок по умолчанию автоматически выполняет полный разведочный анализ: создает Гистограмму и Box plot для каждого числового признака и Столбчатую диаграмму для каждого категориального признака, позволяя переключаться между ними в окне визуализации.

В «Упрощенном режиме» блок автоматически создает несколько типов графиков, позволяя всесторонне оценить данные.



48.5.1. Анализ результатов

С 1. Гистограмма и Box plot (для числовых признаков)

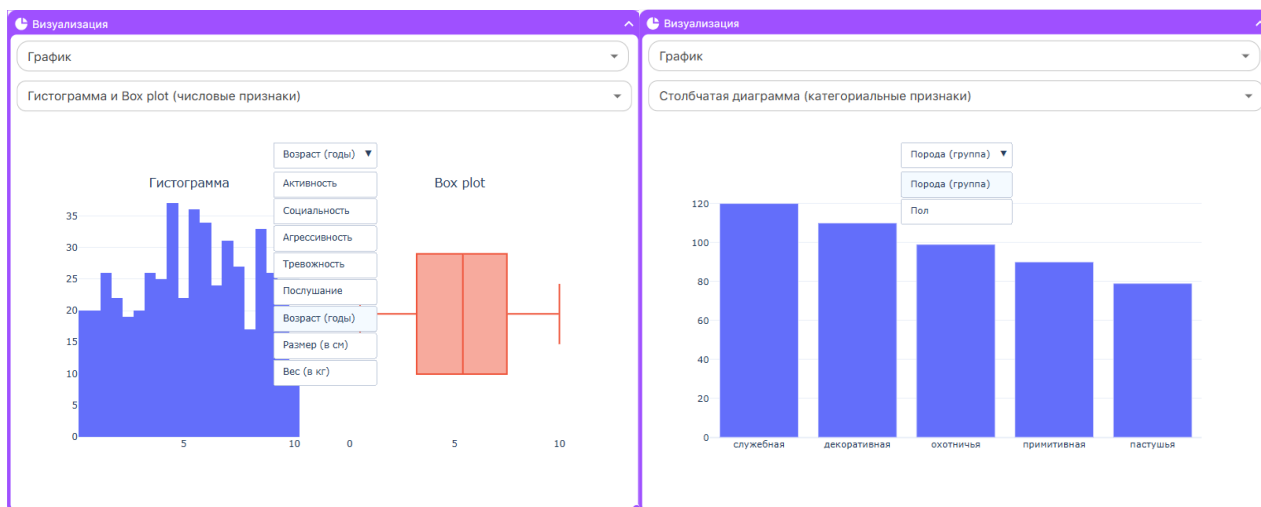
На примере признака «Возраст (годы)»:

- Гистограмма показывает, что большинство собак в выборке имеют возраст от 4 до 8 лет. Собак младше 3 и старше 9 лет заметно меньше.
- Box plot подтверждает это: 50% всех собак имеют возраст примерно от 3 до 7.5 лет. Медианный возраст — около 5.4 года. Аномальных выбросов нет.

48.5.2. Столбчатая диаграмма (для категориальных признаков)

На примере признака «Порода (группа)»:

— Диаграмма показывает количество собак в каждой группе. Наиболее многочисленная — «служебная» (120 собак), наименее — «пастушья» (около 80).



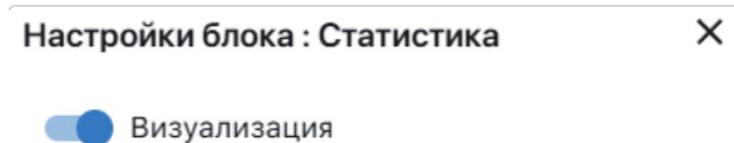
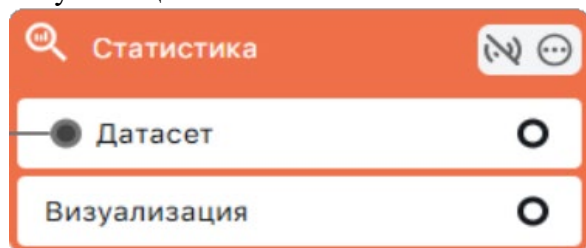
Вывод: упрощенный режим позволяет быстро оценить, что возрастной состав выборки неоднороден (преобладают взрослые собаки), а распределение по породам не сбалансировано.

48.6 Расчет описательных статистик

Рассчитаем основные описательные статистики (среднее, медиану, стандартное отклонение, минимум, максимум и т.д.) для числовых столбцов датасета с помощью блока «Статистика».

Настройка

Блок прост в использовании: достаточно подключить к нему датасет и активировать визуализацию.



11.6.1. Анализ результатов

Анализ таблицы с числовыми характеристиками датасета показывает следующее:

- Размер выборки: Датасет содержит 498 записей. Пропущенные значения в проанализированных столбцах отсутствуют.
- Разброс данных: Наибольшая вариативность наблюдается в физических параметрах. Стандартное отклонение для Размера (15.34) и Веса (7.65) значительно выше, чем для поведенческих черт, например, для Социальности (1.42).
- Диапазоны значений: Все числовые показатели находятся в корректных диапазонах. Минимальный вес составляет 0.2 кг, что является физически возможным значением.
- Средние показатели: Средний балл Послушания (5.65) заметно выше среднего балла Агрессивности (3.68).

Информация по датасету								
index	Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)
Количество значений	498.0	498.0	498.0	498.0	498.0	498.0	498.0	498.0
среднее	5.550200803212851	5.51004016064257	3.682730923694779	4.781124497991968	5.646586345381526	5.3024096385542165	45.08132530120482	15.548995983935741
стандартное отклонение	2.153675899274086	1.4244557004868459	2.5246756015866736	2.0852758553533794	2.281850001677417	2.6629725408625218	15.341739355408768	7.651288932589313
минимум	1.0	3.0	1.0	2.0	2.0	0.5	7.1	0.2
25-й процентиль	5.0	5.0	2.0	4.0	4.0	3.2	33.15	9.825000000000001
медиана (50-й процентиль)	6.0	5.0	3.0	5.0	5.0	5.4	44.45	15.5
75-й процентиль	7.0	5.0	5.0	5.0	8.0	7.5	55.675000000000004	20.975
максимум	8.0	9.0	9.0	9.0	10.0	10.0	86.4	40.5

Вывод: выборка состоит из очень разнообразных по размеру и весу собак, но более однородных в своих поведенческих проявлениях. Данные не содержат очевидных аномалий или пропусков.

Таким образом, разведочный анализ данных (EDA) — это необходимый подготовительный этап, который позволяет оценить качество данных, выявить их особенности и выбрать подходящие методы для дальнейшей обработки и моделирования.

После того как данные исследованы и подготовлены, можно переходить к непосредственному решению практических задач — построению моделей машинного обучения.

49. Примеры обучения моделей

На платформе Машинное обучение модели ИИ происходит посредством сборки пайплайнов — наглядных схем, которые вы конструируете из готовых функциональных блоков. Соединяя блоки друг с другом и настраивая их параметры, вы можете решать различные задачи машинного обучения.

Рассмотрим простейшие примеры пайплайнов для решения трёх классических задач машинного обучения: классификации, регрессии и кластеризации.

49.1 Классификация. Определение вида животного

Задача

Предсказать, к какому из нескольких известных классов относится объект.

Рассмотрим пошаговый сбор пайплайна для решения задачи бинарной классификации.

Цель

Создать модель, которая на основе физических характеристик животного (веса и длины хвоста) сможет предсказать его вид — кошка или собака.

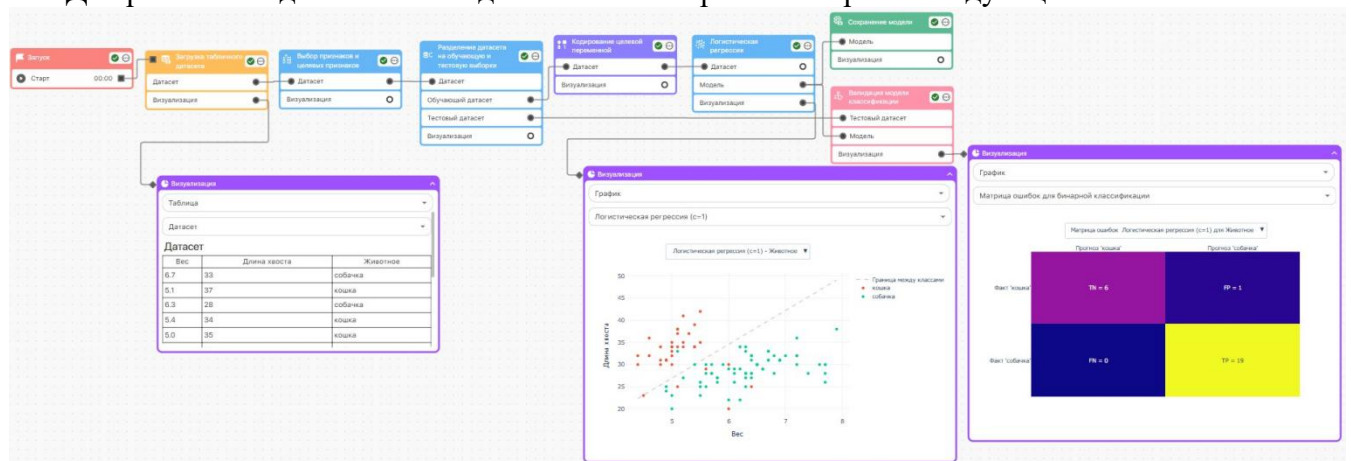
Датасет

В качестве исходных данных используется таблица с тремя столбцами:

- Вес - числовой признак
- Длина хвоста - числовой признак
- Животное - целевая категориальная переменная: 'кошка' или 'собака'

Построение пайплайна

Для решения задачи мы последовательно соберем и настроим следующий пайплайн:

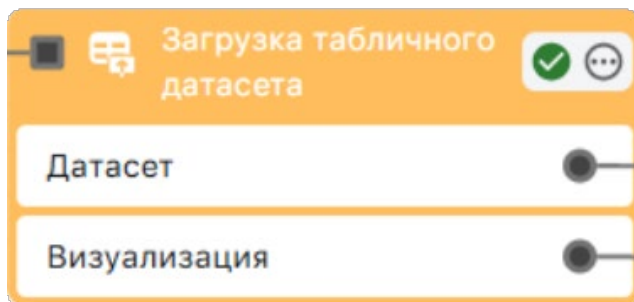


Шаг 1. Загрузка данных

Загрузим заранее созданный табличный датасет с помощью блока «Загрузка табличного датасета».

Настройка

Нажмите на иконку настроек и в выпадающем списке «Выберите датасет для загрузки» укажите нужный файл, в нашем случае — «Кошки и собаки. Тренировочная выборка».



Настройки блока : Загрузка табличного датасета

Выберите датасет для загрузки

Кошки и собаки. Тренировочная ...

☒ Визуализация

Восстановить значения по умолчанию

Ниже представлена визуализация загруженных данных - таблица с тремя столбцами: "Вес", "Длина хвоста" и "Животное".

Визуализация		
Таблица		
Датасет		
Вес	Длина хвоста	Животное
6.7	33	собачка
5.1	37	кошка
6.3	28	собачка
5.4	34	кошка
5.0	35	кошка
6.4	28	собачка
6.4	25	кошка
7.2	32	собачка
5.2	34	кошка
6.7	33	собачка
6.7	30	собачка
6.3	25	собачка
6.5	30	собачка
5.0	30	кошка
6.2	34	собачка

Шаг 2. Выбор признаков

С помощью блока «Выбор признаков и целевых признаков» укажем, на основе каких данных будем обучать модель (признаки) и что именно предсказывать (целевой признак).

Настройка

В поле «Признаки» выберем колонки, которые будут использоваться для обучения: Вес и Длина хвоста.

В поле «Целевые признаки» выберем колонку, которую мы хотим предсказывать: Животное.

Чтобы быстро выбрать все доступные признаки, нажмите справа на иконку «Выгрузить признаки».

Выбор признаков и целевых признаков

Датасет

Визуализация

Настройки блока : Выбор признаков и целевых признаков

Выгрузить признаки
Кошки и собаки. Тренировочная выборка

Признаки
Вес × Длина хвоста ×

Выгрузить признаки
Кошки и собаки. Тренировочная выборка

Целевые признаки
Животное ×

Визуализация

Шаг 3. Разделение выборки

Используя блок «Разделение датасета на обучающую и тестовую выборки», разделим исходный датасет на две части: обучающую (на которой модель будет учиться) и тестовую (на которой мы проверим качество её работы).

Настройка

В настройках блока выберите «Упрощённый режим», который автоматически делит данные. Доля тестовой выборки по умолчанию составляет 0.2 (т.е. 20% данных пойдут на тест, а 80% — на обучение).

Разделение датасета на обучающую и тестовую выборки

Датасет

Обучающий датасет

Тестовый датасет

Визуализация

Настройки блока : Разделение датасета на обучающую и тестовую выборки

☐ Упрощённый режим

Доля тестовой выборки в датасете
0.1 0.5 0.2

☒ Перемешивать наблюдения перед разделением

☐ Разделять с учетом меток классов

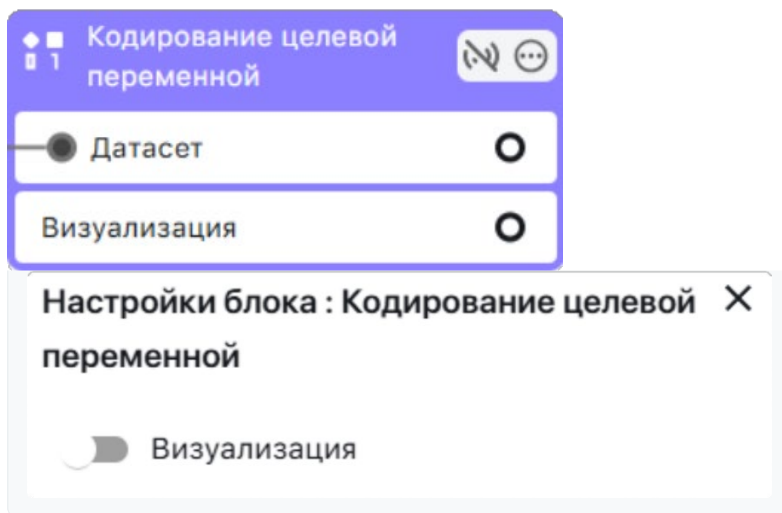
Визуализация

Шаг 4. Кодирование целевой переменной

Логистическая регрессия (алгоритм, который мы будем использовать в этой задаче) требует, чтобы целевая переменная была числовой. Поскольку наша колонка «Животное» является категориальной, преобразуем текстовые метки ('кошка', 'собака') в числовые при помощи блока «Кодирование целевой переменной».

Настройка

Блок не требует настроек. Просто соединяем блоки.

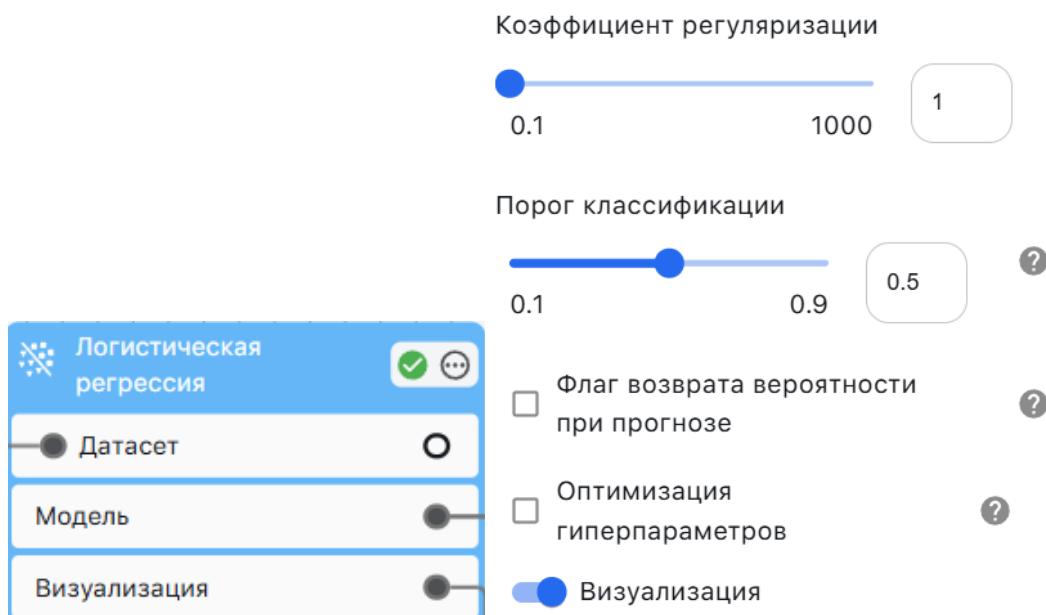


Шаг 5. Обучение модели

Для обучения модели бинарной классификации выберем блок «Логистическая регрессия».

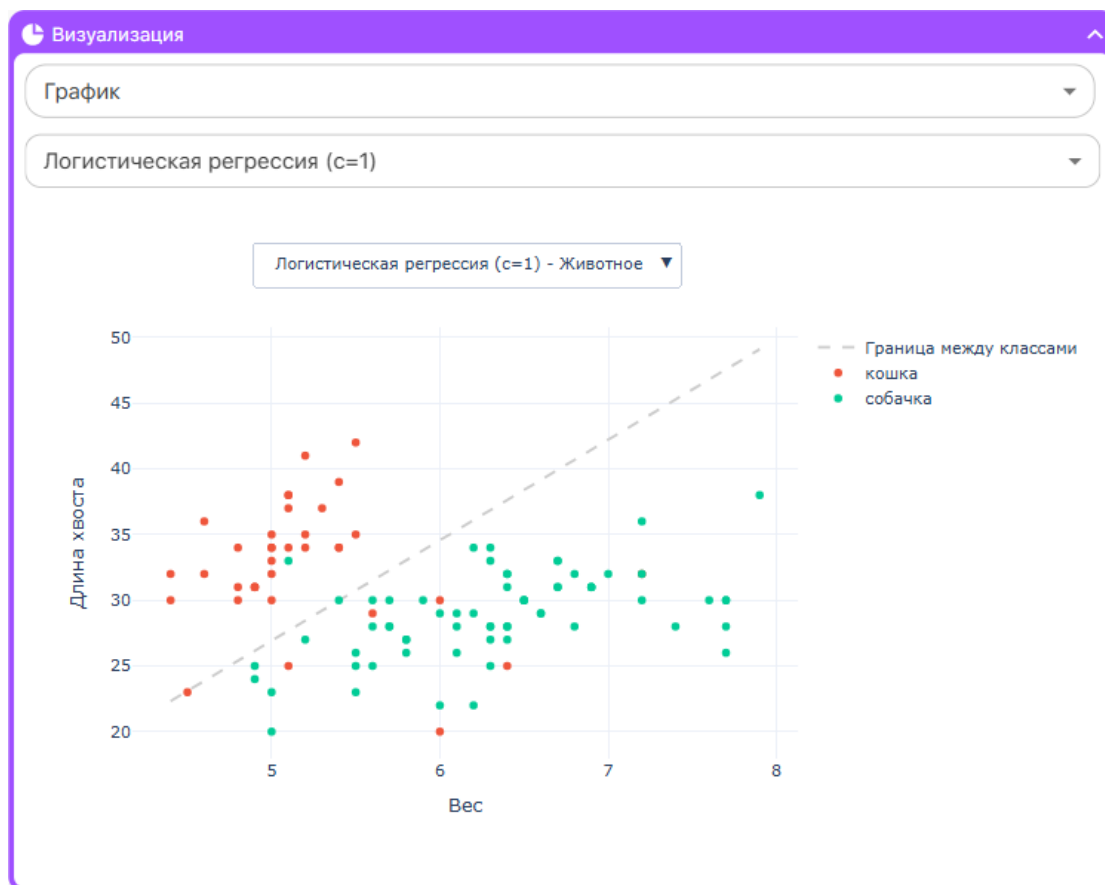
Настройка

Используем «Упрощённый режим», который автоматически применяет стандартные настройки. Чтобы увидеть их, снимите флажок.



На представленной ниже визуализации виден результат обучения: как модель разделила пространство признаков (Вес и Длина хвоста).

Визуально можно оценить, насколько хорошо модель справляется с разделением классов. В нашем случае классы разделены достаточно четко.

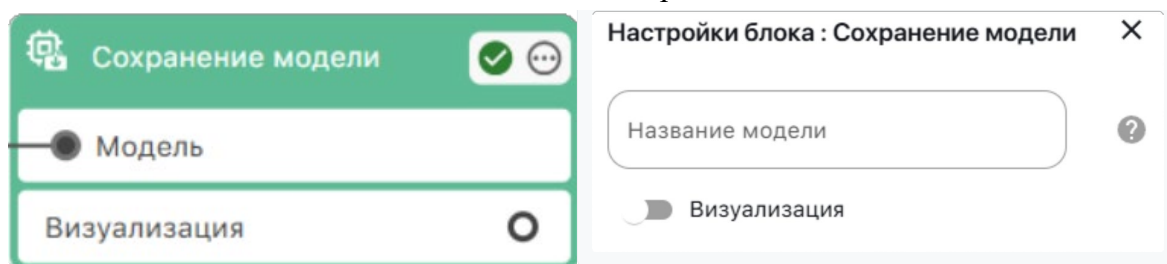


Шаг 6. Сохранение модели

Блок «Сохранение модели» позволяет сохранить обученную модель для будущего использования, например, для предсказаний на новых данных.

Настройка

В поле «Название модели» введите имя для сохранения.

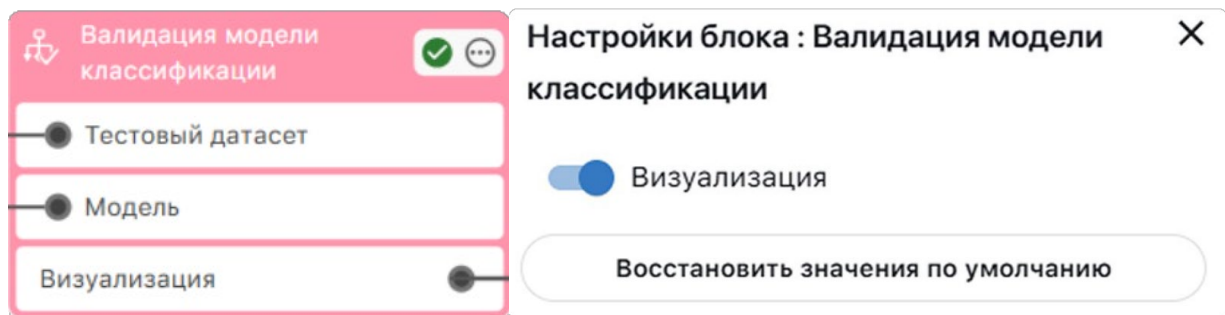


Шаг 7. Валидация модели

На этом шаге мы проверим качество нашей модели. Блок «Валидация модели классификации» берет тестовую (валидационную) выборку, которую модель еще не видела, и сравнивает предсказания с реальными ответами.

Настройка

Блок не требует настроек. Он принимает на вход тестовый датасет и обученную модель, а на выходе предоставляет результаты для визуализации. Просто подключите входы и активируйте визуализацию.



49.2 Анализ результатов

1. Матрица ошибок (Confusion Matrix)

Это ключевой инструмент для оценки качества классификации. Он детально показывает, где модель была права, а где ошиблась.

Интерпретация:

- TN (True Negative) = 6
- TP (True Positive) = 19
- FP (False Positive) = 1
- FN (False Negative) = 0

Вывод: Модель демонстрирует высокую точность. На всей тестовой выборке ($6 + 19 + 1 + 0 = 26$ животных) была допущена всего одна ошибка. Особенно важно, что модель безошибочно находит всех собачек ($FN=0$).

2. График кривой ROC-AUC

Этот график показывает зависимость доли верно классифицированных объектов от доли неверно классифицированных при разных порогах чувствительности модели.

Интерпретация:

Чем ближе кривая к левому верхнему углу, тем лучше модель. Модель настолько уверена в своих прогнозах, что находит всех 19 настоящих собачек (TP Rate взлетает до 1.0), прежде чем совершить свою единственную ошибку, классифицировав одну кошку как собаку (FP Rate сдвигается с 0 до $1/7$).

Вывод: Такой вид — прямое следствие того, что модель очень легко и четко разделила два класса. Площадь под этой кривой (AUC) будет очень близка к 1.0. Это является показателем хорошего классификатора.

3. Таблица метрик

Эта таблица агрегирует ключевые показатели качества модели для быстрой оценки.

- Accuracy = 0.962
- Precision = 0.95
- Recall = 1.0
- F1-score = 0.974

Вывод: Все метрики близки к идеальным значениям, что подтверждает высокое качество и надежность модели.

4. Классификационный отчет (Classification Report)

Этот отчет предоставляет самую детализированную разбивку качества модели, показывая её производительность для каждого класса в отдельности.

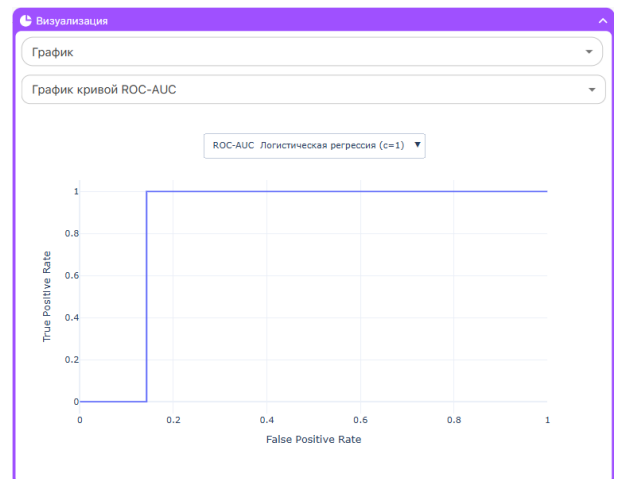
Интерпретация :

Отчет состоит из строк для каждого класса (кошка, собачка) и нескольких итоговых:

Метрика	Класс «кошка»	Класс «собачка»
precision	1.0	0.95
recall	0.86	1.0
f1-score	0.92	0.97
support	7	19

Итоговые метрики:

- accuracy = 0.96: Общая доля правильных ответов по всей выборке.
- macro avg: Среднее арифметическое метрик по всем классам. Каждый класс имеет одинаковый вес.
- weighted avg: Взвешенное среднее, где метрики каждого класса учитываются пропорционально его размеру. Эта метрика лучше отражает общее качество на несбалансированных данных, как в нашем случае (19 собак против 7 кошек).



Визуализация

Таблица

Классификационный отчет Логистическая регрессия (c=1) по Животное

Классификационный отчет Логистическая регрессия (c=1) по Животное

	precision	recall	f1-score	support
кошка	1.0	0.86	0.92	7
собачка	0.95	1.0	0.97	19
accuracy	0.96	0.96	0.96	0
macro avg	0.97	0.93	0.95	26
weighted avg	0.96	0.96	0.96	26

Визуализация

Таблица

Сравнительная таблица

Сравнительная таблица

	Accuracy	F1	Precision	Recall
Логистическая регрессия (c=1)_Животное	0.962	0.974	0.95	1.0

Вывод: классификационный отчет показывает, что модель работает превосходно, но не одинаково для обоих классов. Она идеально находит всех собак (recall=1.0 для собак), но иногда может принять кошку за собаку (recall=0.86 для кошек). При этом, если модель вынесла вердикт "это кошка", она никогда не ошибается (precision=1.0 для кошек).

49.3 Регрессия. Предсказание возраста животного

Задача

Предсказать непрерывное числовое значение на основе набора признаков. В отличие от классификации, где мы предсказываем категорию, здесь мы предсказываем конкретное число.

Цель

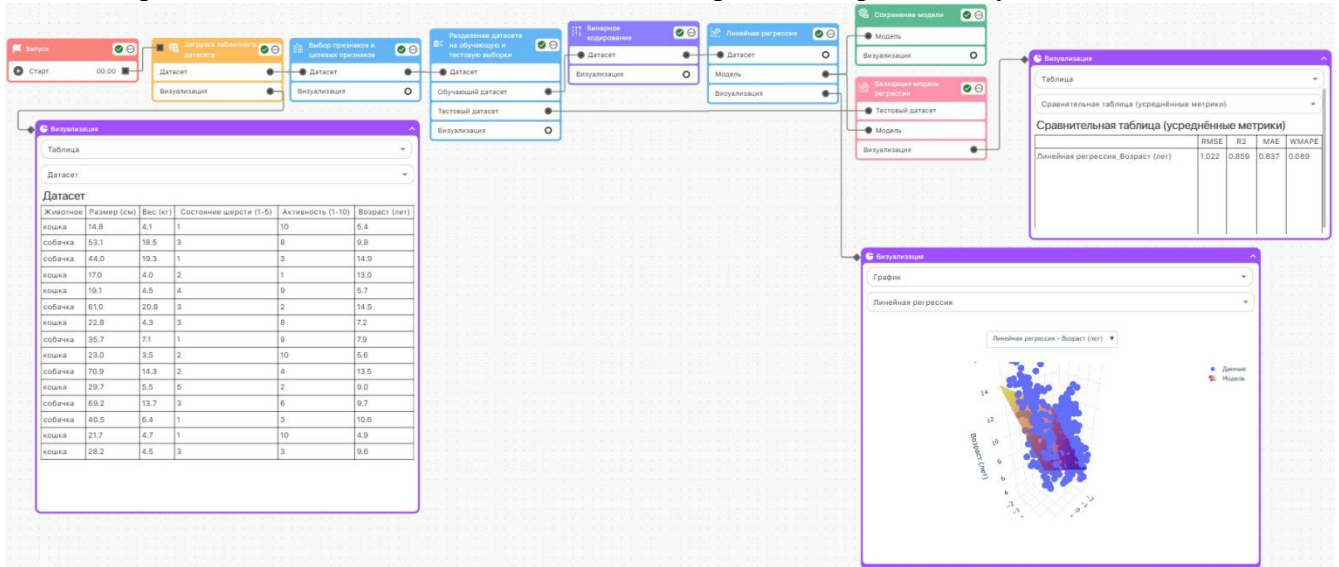
Создать модель, которая на основе физических и поведенческих характеристик животного (Животное, Размер, Вес, Состояние шерсти, Активность) сможет предсказать его возраст.

Датасет

Используется таблица с характеристиками кошек и собак, где целевой переменной является столбец «Возраст (лет)».

Построение пайплайна

Для решения задачи мы последовательно соберем и настроим следующий пайплайн:



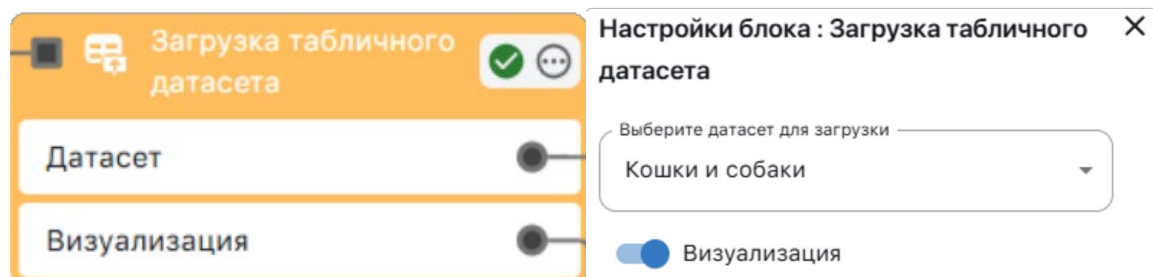
Шаг 1. Загрузка данных

Первый шаг — загрузка данных. Используем блок «Загрузка табличного датасета».

Настройка

В настройках блока в выпадающем списке указываем датасет «Кошки и собачки».

Активируем визуализацию.



Ниже представлена визуализация загруженных данных. Таблица содержит следующие столбцы, которые мы будем использовать для обучения модели:

- Животное — категориальный признак ('кошка' или 'собака').
- Размер (см) и Вес (кг) — физические параметры.
- Состояние шерсти (1-5) и Активность (1-10) — поведенческие и физические оценки.
- Возраст (лет) — целевая переменная, которую мы будем предсказывать.

Визуализация

Таблица

Датасет

Датасет

Животное	Размер (см)	Вес (кг)	Состояние шерсти (1-5)	Активность (1-10)	Возраст (лет)
кошка	14.8	4.1	1	10	5.4
собачка	53.1	18.5	3	8	9.8
собачка	44.0	19.3	1	3	14.9
кошка	17.0	4.0	2	1	13.0
кошка	19.1	4.5	4	9	5.7
собачка	61.0	20.8	3	2	14.5
кошка	22.8	4.3	3	8	7.2
собачка	35.7	7.1	1	9	7.9
кошка	23.0	3.5	2	10	5.6
собачка	70.9	14.3	2	4	13.5
кошка	29.7	5.5	5	2	9.0
собачка	69.2	13.7	3	6	9.7
собачка	40.5	6.4	1	3	10.6
кошка	21.7	4.7	1	10	4.9
кошка	28.2	4.5	3	3	9.6

Шаг 2. Выбор признаков

Укажем, на основе каких данных будем обучать модель и что именно предсказывать с помощью блока «Выбор признаков и целевых признаков».

Настройка

В поле «Признаки» выбираем все колонки, которые будут использоваться для предсказания: Животное, Размер (см), Вес (кг), Состояние шерсти (1-5) и Активность (1-10).

В поле «Целевые признаки» выбираем колонку, которую хотим предсказать: Возраст (лет).

Совет: чтобы быстро выбрать все доступные признаки, нажмите справа на иконку Выгрузить признаки».

The image shows a block titled 'Разделение датасета на обучающую и тестовую выборки' (Splitting the dataset into training and testing sets) and its settings panel. The block has four input fields: 'Датасет' (Dataset), 'Обучающий датасет' (Training dataset), 'Тестовый датасет' (Testing dataset), and 'Визуализация' (Visualization). The settings panel, titled 'Настройки блока : Выбор признаков и целевых признаков' (Block settings: Feature and target feature selection), includes two dropdowns for 'Выгрузить признаки' (Export features) and 'Целевые признаки' (Target features), both set to 'Кошки и собаки' (Cats and dogs). It also has a list of selected features: 'Животное' (Animal), 'Размер (см)' (Size (cm)), 'Вес (в кг)' (Weight (kg)), 'Состояние шерсти (1-5)' (Fur condition (1-5)), and 'Активность (1-10)' (Activity (1-10)). A 'Визуализация' (Visualization) toggle is at the bottom.

Шаг 3. Разделение выборки

С помощью блока «Разделение датасета на обучающую и тестовую выборки», делим исходный набор данных на две части: обучающую, на которой модель будет искать закономерности, и валидационную, на которой мы проверим качество её предсказаний.

Настройка

Оставляем «Упрощённый режим». Он автоматически выполняет разделение в стандартной пропорции 80/20.

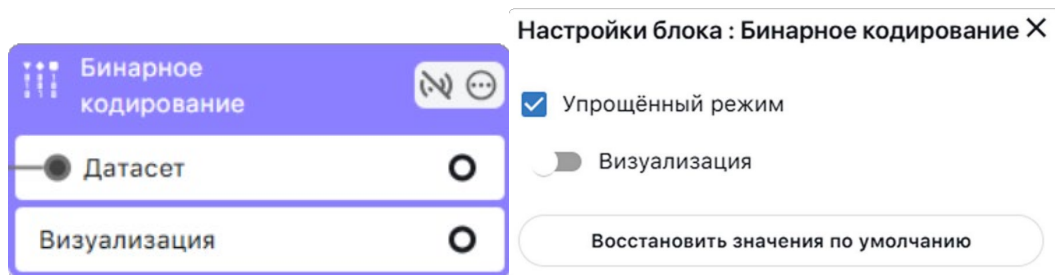
The image shows the same block and settings panel as above, but with the 'Упрощённый режим' (Simplified mode) checkbox checked. The 'Визуализация' (Visualization) toggle remains unchecked.

Шаг 4. Бинарное кодирование

Линейная регрессия, как и многие другие модели, работает только с числовыми данными. Поскольку признак «Животное» является категориальным ('кошка', 'собака'), необходимо преобразовать его в числовой формат. Используем блок «Бинарное кодирование».

Настройка

Оставляем «Упрощённый режим», который автоматически находит и кодирует все категориальные признаки в датасете.

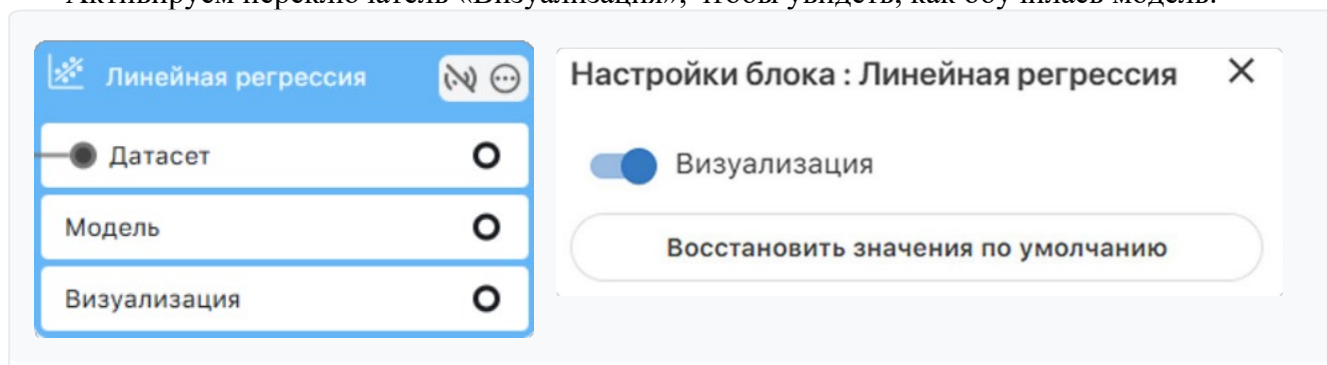


Шаг 5. Обучение модели

Для решения задачи выберем алгоритм — «Линейная регрессия». Он попытается найти линейную зависимость между признаками и возрастом животного.

Настройка

Активируем переключатель «Визуализация», чтобы увидеть, как обучилась модель.



Анализ результата обучения

На представленной ниже визуализации модель пытается провести гиперплоскость через многомерное пространство признаков, чтобы наилучшим образом соответствовать реальному возрасту. Поскольку у нас 5 признаков, для наглядности с помощью метода главных компонент (PCA) они сжимаются в два измерения (PCA1, PCA2), чтобы можно было построить 3D-график.

1. Данные делятся на два кластера. Видно два четких "облака" точек. Это визуальное отражение самого сильного признака в данных — типа животного. В одном облаке сгруппированы кошки, в другом — собаки.
2. Модель успешно справилась с двумя популяциями. Плоскость предсказаний не "провалилась" в один кластер, игнорируя второй. Она уверенно проходит через оба, находя общую закономерность.
3. Высокая точность. Большинство точек расположены близко к плоскости предсказаний. Это означает, что для большинства животных прогноз модели близок к реальному возрасту. Вертикальное расстояние от точки до плоскости — наглядная ошибка предсказания.



Вывод: модель успешно справилась с анализом двух разных групп. Плоскость предсказаний проходит не через одну группу, игнорируя вторую, а уверенно располагается между ними, описывая общую зависимость. Это говорит о её высокой обобщающей способности.

Шаг 6. Сохранение модели

Блок «Сохранение модели» позволяет сохранить обученную модель для будущего использования, например, для предсказаний на новых данных.

Настройка

В поле «Название модели» введите имя для сохранения.

The screenshot shows the configuration interface for the 'Сохранение модели' (Save Model) block. On the left, there is a green block icon with a gear, a checkmark, and a menu icon. Below it, there are two tabs: 'Модель' (Model) and 'Визуализация' (Visualization). The 'Модель' tab is selected. On the right, there is a panel titled 'Настройки блока : Сохранение модели' (Block settings : Save model) with a close button. Inside this panel, there is a text input field labeled 'Название модели' (Model name) with a question mark icon to its right. Below the input field, there is a toggle switch labeled 'Визуализация' (Visualization) which is currently turned off.

Шаг 7. Валидация модели

На этом шаге мы проверим качество нашей модели с помощью блока «Валидация модели регрессии». Он берет тестовую выборку, которую модель еще не видела, сравнивает предсказания с реальными ответами и рассчитывает метрики качества.

Настройка

Блок не требует настроек. Просто подключите на вход тестовый датасет и обученную модель, затем в настройках блока активируйте визуализацию.

Вывод: все метрики указывают на высокое качество построенной модели. Она хорошо объясняет данные (высокий R^2) и имеет низкую, приемлемую ошибку предсказания (в среднем меньше года). Модель готова к использованию.

49.4 Кластеризация. Поиск скрытых групп

Задача

Разделить данные на группы (кластеры) без заранее известных ответов. Алгоритм сам находит объекты, похожие друг на друга.

Цель

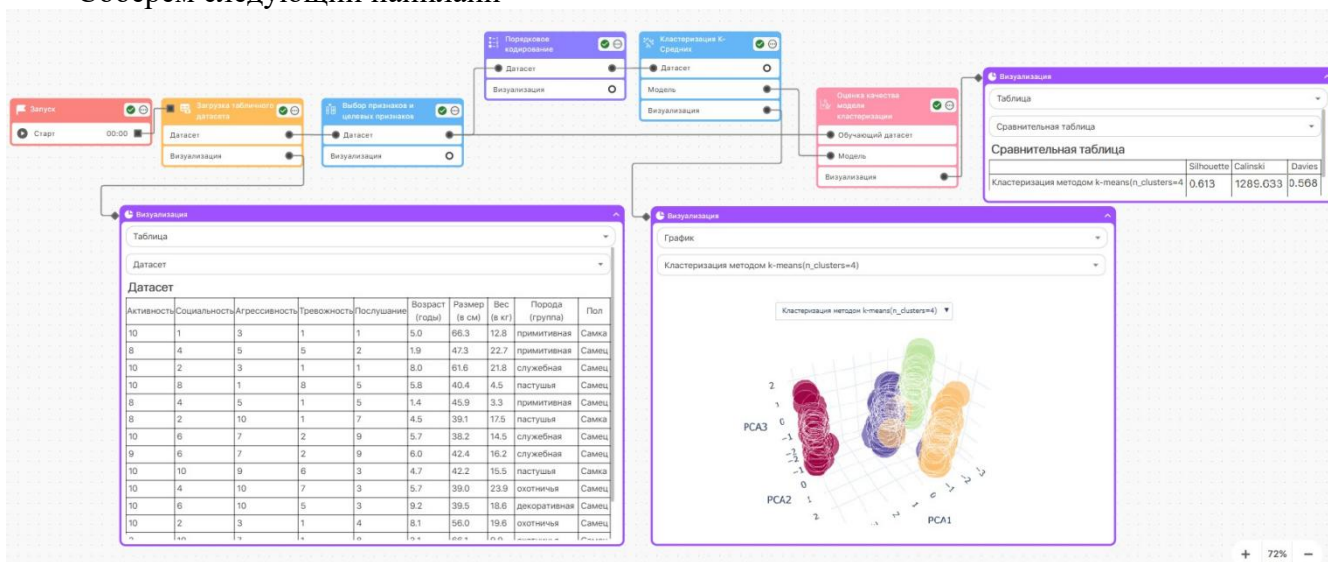
Проверить гипотезу: сможет ли алгоритм, анализируя все данные о собаках (поведенческие, физические, порода), самостоятельно разделить их на 4 группы, которые будут соответствовать их реальным группам пород.

Датасет

Используется таблица с 10 колонками, описывающими поведенческие и физические характеристики собак, а также их пол и группу породы.

Построение пайплайна

Соберем следующий пайплайн



Шаг 1. Загрузка данных

Начинаем с загрузки данных с помощью блока «Загрузка табличного датасета».

Настройка

В настройках указываем датасет «Поведение собак» и активируем визуализацию.

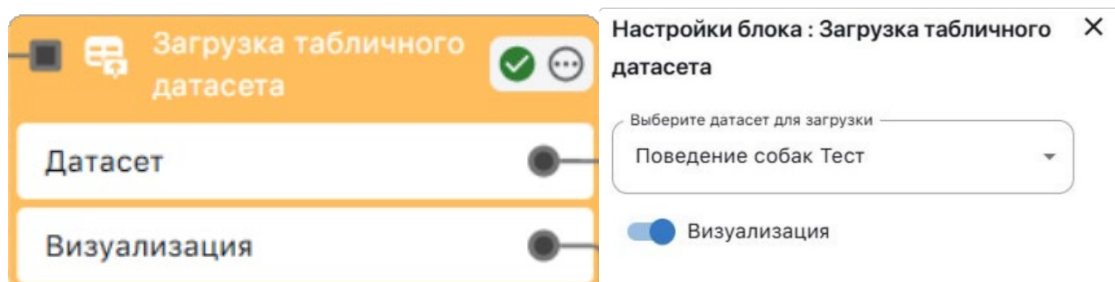


Таблица содержит следующие столбцы, которые мы будем использовать для поиска скрытых групп в данных:

— Активность, Социальность, Агрессивность, Тревожность, Послушание — поведенческие оценки (1-10).

- Возраст, Размер, Вес — физические параметры.
- Порода (группа), Пол — категориальные признаки.

Визуализация									
Таблица									
Датасет									
Датасет									
Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)	Порода (группа)	Пол
10	5	9	6	5	5.7	59.0	23.9	охотничья	Самец
2	9	2	2	2	7.8	23.4	5.1	декоративная	Самец
7	8	4	4	8	3.3	43.4	16.3	пастушья	Самка
9	3	3	2	10	8.0	68.6	31.8	служебная	Самец
2	9	2	2	2	4.1	25.3	6.4	декоративная	Самка
10	5	9	6	5	8.1	56.0	22.6	охотничья	Самец
9	3	3	2	10	1.8	65.8	29.5	служебная	Самец
7	8	4	4	8	5.8	40.4	14.5	пастушья	Самец
10	5	9	6	5	2.3	56.9	21.9	охотничья	Самец
9	3	3	2	10	6.0	67.4	31.2	служебная	Самец
7	8	4	4	8	2.9	44.3	16.0	пастушья	Самец
2	9	2	2	2	5.9	24.7	6.6	декоративная	Самец
7	8	4	4	8	4.7	42.2	15.5	пастушья	Самка
10	5	9	6	5	3.8	58.0	23.0	охотничья	Самец
2	9	2	2	2	9.2	24.5	5.6	декоративная	Самец

Шаг 2. Выбор признаков

В задачах кластеризации мы не разделяем данные на признаки и цель. Передаем все доступные данные, чтобы алгоритм нашел в них скрытые закономерности. Используем блок «Выбор признаков и целевых признаков».

Настройка

В поле «Признаки» выбираем все столбцы. Поле «Целевые признаки» оставляем пустым.

Разделение датасета на обучающую и тестовую выборки

Датасет

Обучающий датасет

Тестовый датасет

Визуализация

Настройки блока : Выбор признаков и целевых признаков

Выгрузить признаки

Поведение собак

Признаки

Активность

Социальность

Агрессивность

Тревожность

Послушание

Порода (группа)

Пол

Возраст (годы)

Размер (в см)

Вес (в кг)

Выгрузить признаки

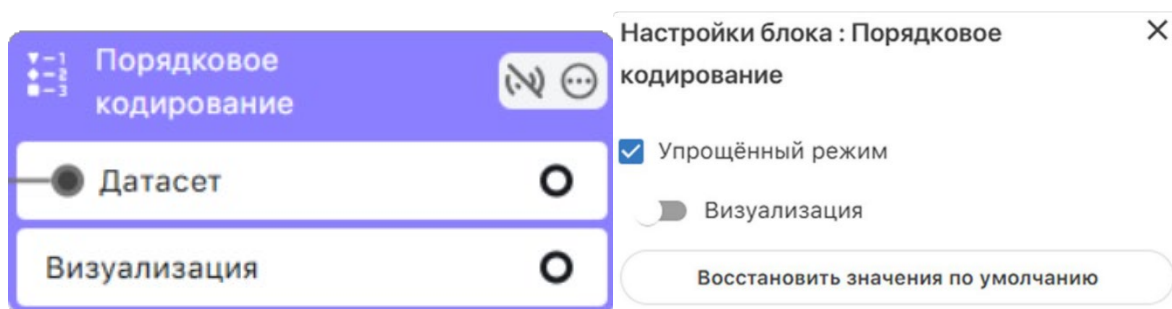
Целевые признаки

Шаг 3. Подготовка данных: Кодирование

Алгоритмы кластеризации, такие как K-Means, работают только с числами. Датасет содержит категориальные столбцы (Порода (группа), Пол), которые необходимо преобразовать в числовой формат. Для этого используем блок «Порядковое кодирование»

Настройка

Оставляем «Упрощённый режим». Он автоматически найдет все категориальные признаки и заменит их на числовые эквиваленты.

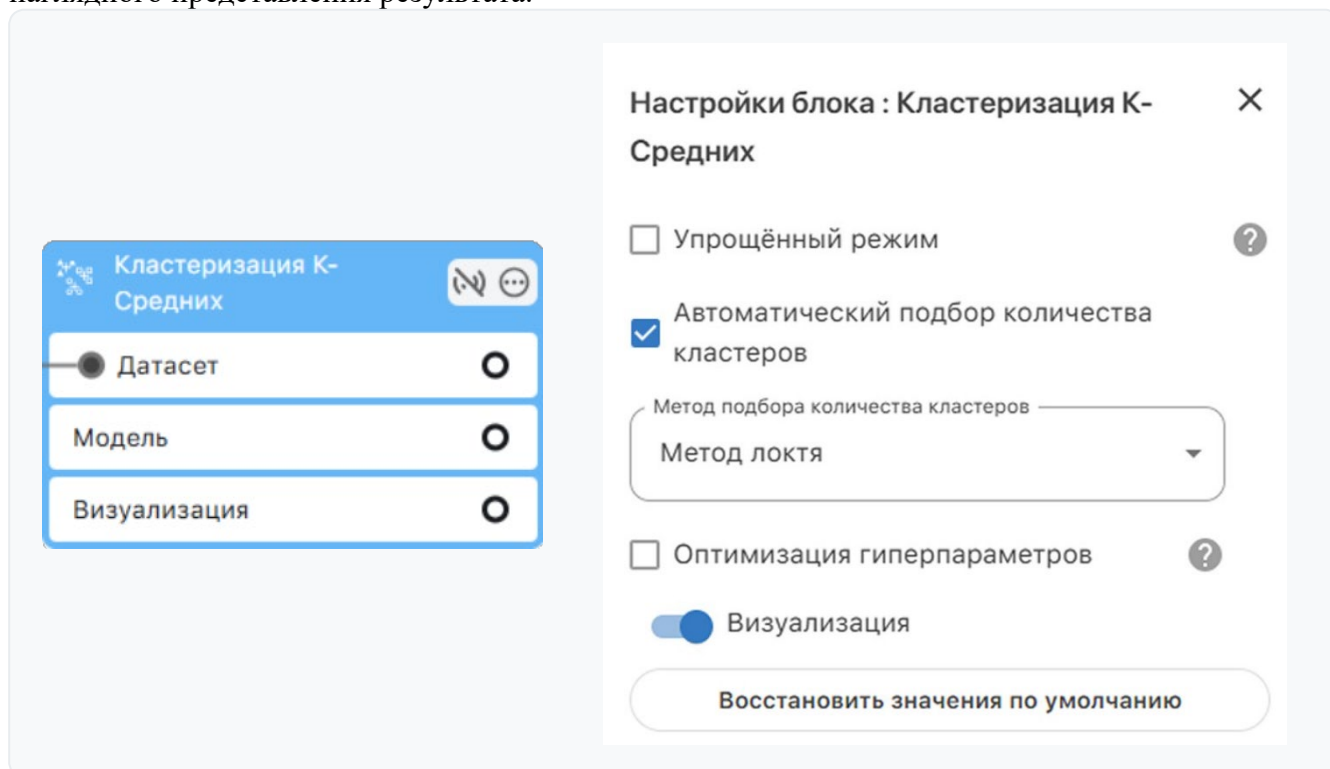


Шаг 4. Обучение модели. Поиск кластеров

Используем блок «Кластеризация К-Средних».

Настройка

Ставим галочку «Автоматический подбор количества кластеров» и выбираем «Метод локтя». Этот метод автоматически протестирует разное количество кластеров (от 2 до 20) и найдет наиболее "естественное" их число для датасета. Активируем визуализацию для наглядного представления результата.



Проанализируем результаты, которые выдал алгоритм после автоматического подбора оптимального количества кластеров.

1. График оценки (Метод локтя)

На графике блок автоматически определил оптимальное количество кластеров равным 4.

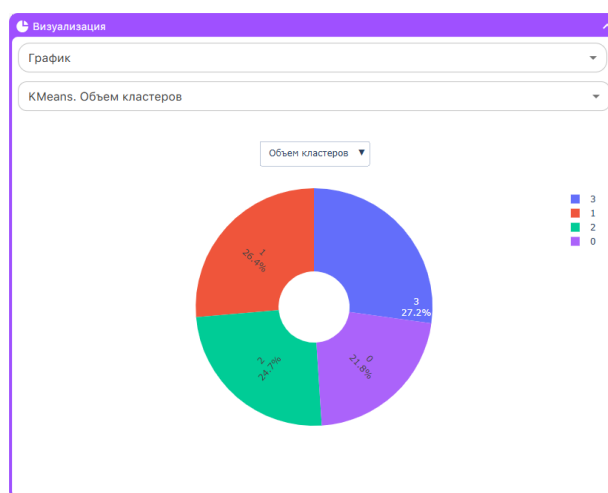
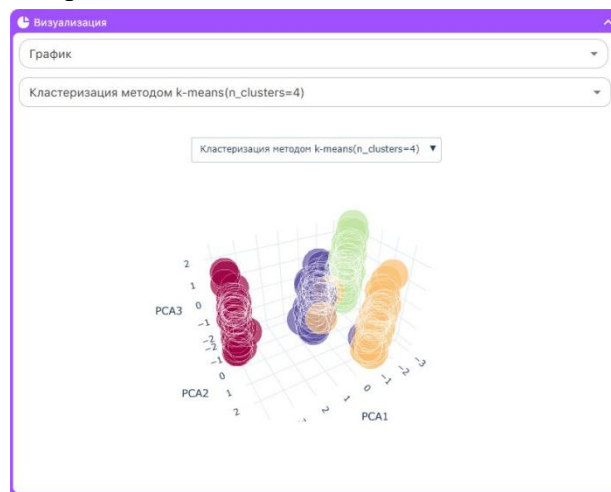
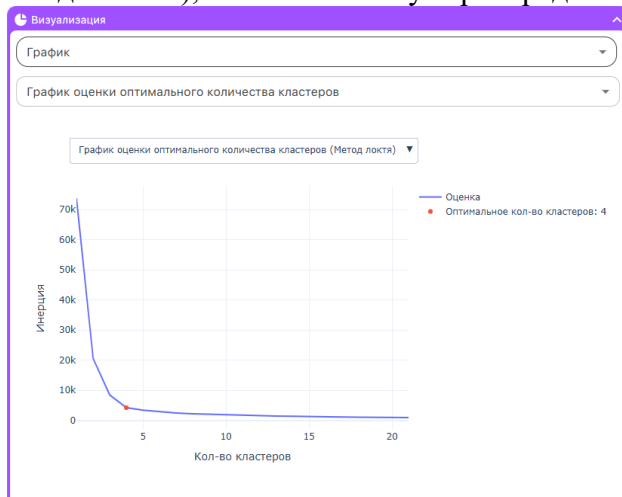
Результат совпадает с реальным количеством пород в исходном датасете.

2. 3D-визуализация кластеров

После того как оптимальное число кластеров было найдено, модель разделила данные на 4 группы. На 3D-графике мы видим результат: четыре очень плотных и четко разделенных облака точек.

3. Объем кластеров (круговая диаграмма)

Диаграмма показывает, что 4 найденных кластера имеют сбалансированные размеры (от 21.8% до 27.2%), что соответствует распределению пород в данных.



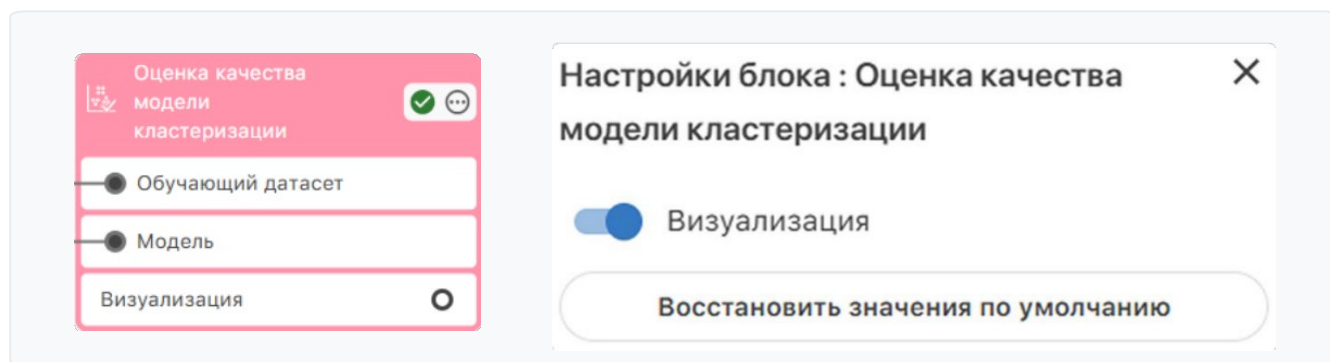
Вывод: кластеризация прошла успешно. Алгоритм определил оптимальное количество кластеров равным четырем, что полностью совпадает с числом реальных породных групп (служебные, пастушьи, охотничьи, декоративные) в данных. Это доказывает, что модель корректно сгруппировала собак по их естественным типам, опираясь на поведенческие и физические признаки.

Шаг 5. Оценка качества кластеризации

Блок «Оценка качества модели кластеризации» рассчитывает числовые показатели, которые говорят, насколько «удачными» получились кластеры.

Настройка

Блок не требует настроек, достаточно просто включить визуализацию.



Проанализируем результаты, чтобы дать объективную оценку модели.

1. Метрики качества кластеризации

Сводная таблица метрик предоставляет количественную оценку качества разделения на кластеры:

Коэффициент силуэта (Silhouette) = 0.613: Достаточно высокое значение, указывает на то, что объекты внутри кластеров очень похожи, а сами кластеры при этом хорошо отделены друг от друга.

Индекс Калински-Харабас (Calinski) = 1289.6: Кластеры получились плотными и четко разграниченными.

Индекс Дэвиса-Болдина (Davies) = 0.568: Низкое значение этого индекса является еще одним подтверждением высокого качества кластеризации.

2. Статистика признаков по кластерам

Таблица позволяет нам «заглянуть внутрь» каждого кластера и понять, какой именно профиль собак в него попал:

- Кластер 0 (пастушьи): характеризуется высокой социальностью (8) и послушанием (8) при средней активности (7). Это соответствует профилю пастушьей собаки — умной, обучаемой и ориентированной на человека.
- Кластер 1 (декоративные): отличается очень малыми размерами (26 см, 7 кг), низкой активностью (2) и послушанием (2), но высокой социальностью (9). Это типичный портрет компаньона.
- Кластер 2 (служебные): крупные собаки (65 см, 30 кг) с максимальным послушанием (10) и активностью (9), но низкой социальностью (3). Профиль идеально описывает рабочую собаку, сконцентрированную на задаче, а не на общении.
- Кластер 3 (охотничьи): самые активные (10) и «агрессивные» (9, что можно трактовать как охотничий инстинкт) собаки со средними показателями послушания и социальности.

Визуализация			
Таблица			
Сравнительная таблица			
Сравнительная таблица			
	Silhouette	Calinski	Davies
Кластеризация методом k-means(n_clusters=4)	0.613	1289.633	0.568

Визуализация		
Таблица		
Количество объектов в каждом кластере для модели KMeans		
Количество объектов в каждом кластере для модели KMeans		
Метка кластера	Количество наблюдений в кластере	
0	0	65
1	1	52
2	2	59
3	3	63

Визуализация

Таблица

Статистика признаков по кластерам для модели KMeans

Статистика признаков по кластерам для модели KMeans

	Метка кластера	Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)	Порода (группа)	Пол
0	0	7	8	4	4	8	5	49	17	пастушья	Самец
1	1	2	9	2	2	2	5	26	7	декоративная	Самка
2	2	9	3	3	2	10	5	65	30	служебная	Самка
3	3	10	5	9	6	5	5	56	23	охотничья	Самец

Вывод: валидация модели подтверждает ее высочайшее качество и практическую ценность. Количественные метрики (Silhouette, Calinski, Davies) демонстрируют статистически сильный результат. Качественный анализ центроидов показывает, что алгоритм не просто разделил данные, а выделил осмысленные, реальные группы, полностью соответствующие четырем породным типам. Сбалансированность кластеров по количеству объектов (от 52 до 65) дополнительно подтверждает устойчивость полученного решения.

50. Примеры сложных пайплайнов

В этом разделе мы рассмотрим построение сложных пайплайнов для решения задач классификации, регрессии и кластеризации.

50.1 Классификация. Определение источника сообщения

Задача

Определить класс объекта на основе его текстового описания. В нашем случае — это задача бинарной классификации, где по содержанию сообщения нужно определить его источник.

Цель

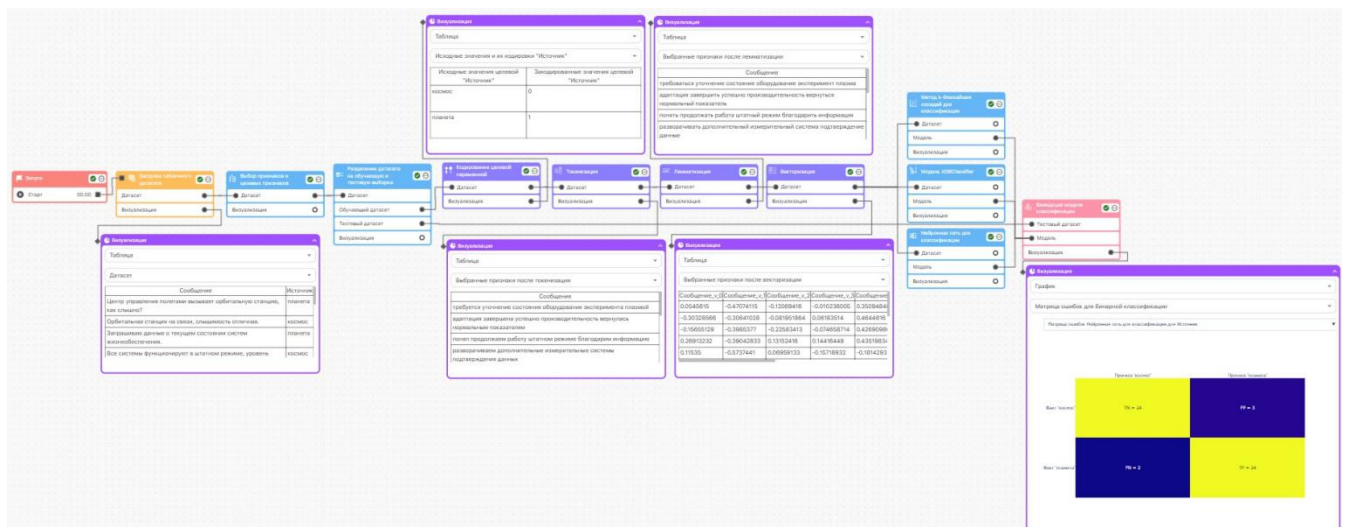
Создать и сравнить несколько моделей, которые смогут по тексту сообщения определить его отправителя: «космос» (Центр управления полетами) или «планета» (Орбитальная станция). В конце мы выберем и сохраним лучшую модель.

Датасет

Используется таблица с перепиской, содержащая два столбца: «Сообщение» (текст) и «Источник» (целевая метка класса).

Построение пайплайна

Для решения задачи последовательно соберем и настроим следующий пайплайн, включающий предобработку текста и параллельное обучение трех разных моделей:



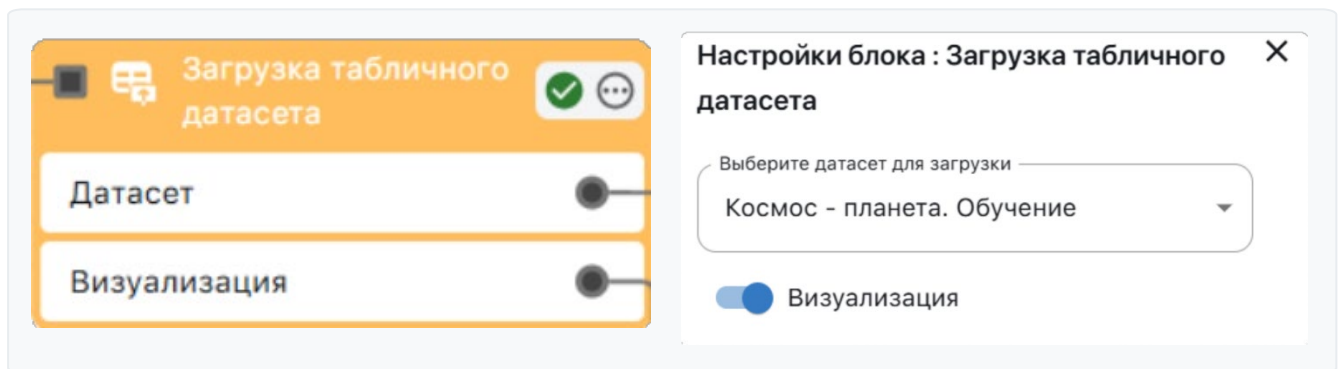
Шаг 1. Загрузка данных

Первый шаг — загрузка данных. Используем блок «Загрузка табличного датасета».

Настройка

В настройках блока в выпадающем списке указываем датасет «Космос - планета.

Обучение». Активируем визуализацию.



Ниже представлена визуализация загруженных данных. Таблица содержит два ключевых столбца:

- Сообщение — текстовый признак, на основе которого будет производиться классификация.
- Источник — целевая переменная ('космос' или 'планета'), которую мы будем предсказывать.

Визуализация

Таблица

Датасет

Сообщение	Источник
Центр управления полетами вызывает орбитальную станцию, как слышно?	планета
Орбитальная станция на связи, слышимость отличная.	космос
Запрашиваю данные о текущем состоянии систем жизнеобеспечения.	планета
Все системы функционируют в штатном режиме, уровень кислорода 98%.	космос
Когда планируете следующую коррекцию орбиты?	планета
Коррекция запланирована через 4 часа 30 минут по бортовому времени.	космос
Получили ли вы последний пакет научных данных?	планета
Подтверждаю получение пакета данных, объем - 2.4 терабайта.	космос
Как обстановка на борту после вчерашнего выхода в открытый космос?	планета
Экипаж восстанавливается согласно графику, показатели в норме.	космос
Наблюдаем возможность солнечной вспышки в вашем секторе.	планета
Принято, активируем дополнительную защиту и перемещаем чувствительное оборудование.	космос
Какова готовность к приему грузового корабля?	планета
Стыковочный модуль подготовлен, системы навигации откалиброваны.	космос
Прогноз погоды на посадочной площадке - переменная облачность.	планета

Шаг 2. Выбор признаков

Укажем, на основе каких данных будем обучать модель и что именно предсказывать с помощью блока «Выбор признаков и целевых признаков».

Настройка

В поле «Признаки» выбираем колонку «Сообщение».

В поле «Целевые признаки» выбираем «Источник».

The image shows a software interface for feature selection. On the left is a block titled 'Разделение датасета на обучающую и тестовую выборки' (Splitting the dataset into training and testing sets). It has a 'Датасет' (Dataset) input field and three output fields: 'Обучающий датасет' (Training dataset), 'Тестовый датасет' (Testing dataset), and 'Визуализация' (Visualization). On the right is a settings panel titled 'Настройки блока : Выбор признаков и целевых признаков' (Block settings: Feature and target feature selection). It contains four sections: 'Выгрузить признаки' (Export features) with a dropdown set to 'Космос - планета. Обучение' (Space - planet. Training); 'Признаки' (Features) with a button 'Сообщение' (Message); another 'Выгрузить признаки' section with a dropdown set to 'Космос - планета. Обучение'; and 'Целевые признаки' (Target features) with a dropdown set to 'Источник' (Source). At the bottom of the settings panel is a 'Визуализация' (Visualization) toggle switch.

Шаг 3. Разделение выборки

С помощью блока «Разделение датасета на обучающую и тестовую выборки» делим исходный набор данных на две части: обучающую, на которой модели будут учиться, и тестовую, на которой проверим качество их работы.

Настройка

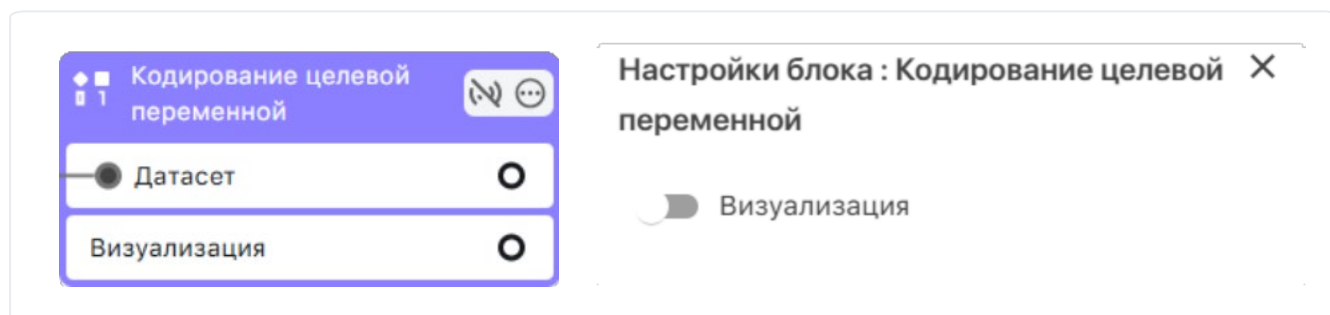
Оставляем «Упрощённый режим». Он автоматически выполняет разделение в стандартной пропорции 80/20.

The image shows the settings for the 'Разделение датасета на обучающую и тестовую выборки' block. The settings panel is titled 'Настройки блока : Разделение датасета на обучающую и тестовую выборки'. It includes a checked checkbox for 'Упрощённый режим' (Simplified mode) and a 'Визуализация' (Visualization) toggle switch.

Шаг 4. Кодирование целевой переменной

Так как алгоритмы машинного обучения работают с числами, а целевая переменная

«Источник» содержит текстовые метки ('космос', 'планета'), преобразуем их в числовой формат с помощью блока «Кодирование целевой переменной».

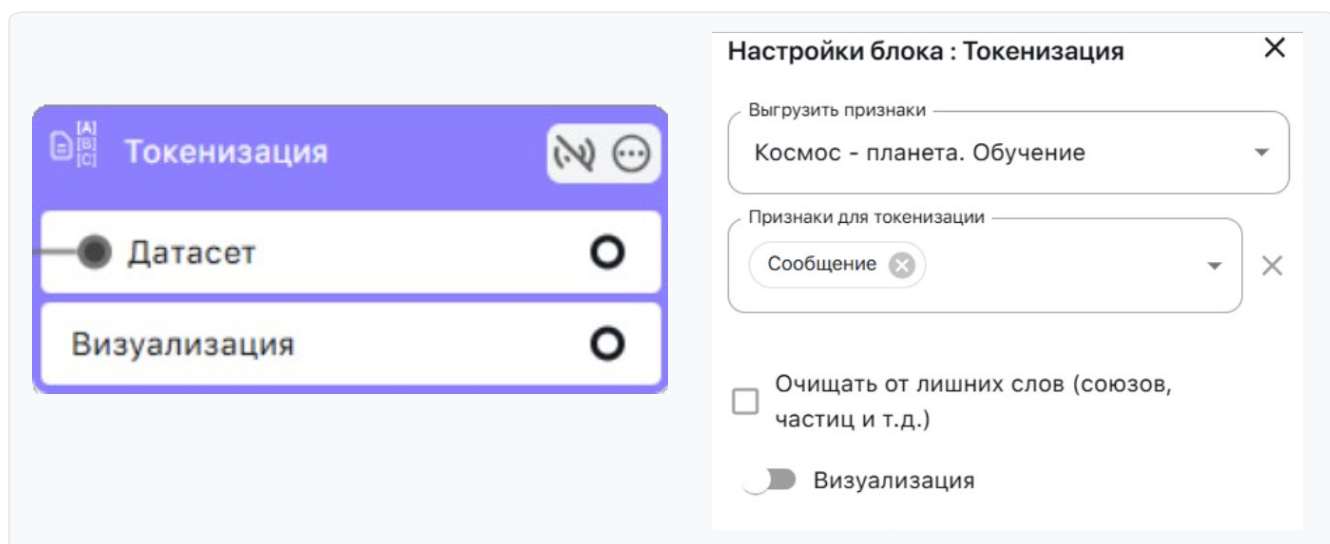


Шаг 5. Предобработка текста: Токенизация

Первый этап обработки текста — Токенизация. Блок «Токенизация» разбивает сплошной текст каждого сообщения на отдельные слова (токены).

Настройка

В настройках блока указываем признак для токенизации — «Сообщение». Также активируем опцию «Очищать от лишних слов (союзов, частиц и т.д.)», чтобы удалить знаки препинания и стоп-слова, которые не несут смысловой нагрузки.



Шаг 6. Предобработка текста: Лемматизация

Следующий шаг — Лемматизация. Блок «Лемматизация» приводит каждое слово к его начальной форме. Например, слова «станции», «станцию», «станцией» будут сведены к единой лемме — «станция». Это необходимо для создания чистого и компактного словаря признаков.

Настройка

Выбираем признак «Сообщение» и оставляем модуль `ru morphology` по умолчанию. Активируем визуализацию, чтобы увидеть результат.

ABC Лемматизация
🔊 ⋮

Датасет
○

Визуализация
○

Настройки блока : Лемматизация ✕

Выгрузить признаки

Космос - планета. Обучение
▼

Признаки для лемматизации

Сообщение ✕
▼
✕

Модуль лемматизации

rumorphy
▼

☒ Визуализация

Анализ результата предобработки

Визуализация показывает результат работы двух предыдущих шагов. В столбце «Сообщение» теперь находятся не целые предложения, а списки очищенных слов, приведенных к их начальной форме (лемме). Например, фраза "требовалось уточнение состояния" теперь представлена как "требоваться уточнение состояние".

Визуализация	
Таблица	
Выбранные признаки после лемматизации	
Сообщение	
требоваться уточнение состояние оборудование эксперимент плазма	
адаптация завершить успешно производительность вернуться	
нормальный показатель	
понять продолжать работа штатный режим благодарить информация	
разворачивать дополнительный измерительный система подтверждение	
данные	
наблюдать аномалия показание внешний датчик температура	
наблюдать небольшой отклонение траектория ваш полёт	
новый расписание принять информация довести член экипаж	
наблюдать аномальный поведение спутник связь соседний орбита	
направлять обновить протокол эксперимент квантовый телепортация	
проверка выполнить герметичность индикатор зелёный	
запрашивать статус готовность эксперимент магнитная левитация	
орбитальный станция связь слышимость отличный	
требоваться подтверждение использование резервный канал связь	
проверка завершить система функционировать штатный режим	
зафиксировать небольшой смещение расписание сеанс связь	

Шаг 7. Векторизация текста

Чтобы модели могли работать с текстом, его нужно преобразовать в числа. Блок «Векторизация» превращает списки слов в числовые векторы.

Настройка

Тип векторизации: Word2Vec. Это мощный алгоритм, который учитывает семантическую близость слов.

Минимальная частота слов: 1. Учитываем все слова. Размер вектора: 300. Каждое сообщение будет представлено вектором из 300 чисел. Это стандартное значение, обеспечивающее хороший баланс между детализацией и производительностью.

Настройки блока : Векторизация

Выгрузить признаки
Космос - планета. Обучение

Признаки для векторизации
Сообщение

☐ Упрощённый режим

Тип векторизации
word2vec

Минимальная частота слов
1 20 1

Размер вектора
10 300 300

☒ Визуализация

Шаг 8. Параллельное обучение трех моделей

Подаем подготовленные данные на вход сразу трем разным алгоритмам классификации, чтобы выяснить, какой из них лучше справится с задачей.

- Метод k-ближайших соседей (KNN): простой и интуитивный алгоритм.
- Модель XGBoost: мощный и популярный алгоритм градиентного бустинга.
- Нейронная сеть для классификации: гибкая модель, способная улавливать сложные зависимости.

Метод k-ближайших соседей для классификации

Датасет

Модель

Визуализация

Настройки блока : Метод k-ближайших соседей для классификации

Количество ближайших соседей

2 200 5

Тип веса для соседей

Единый

Метрика расстояния

Евклидово

Порог классификации

0.1 0.9 0.5

☐ Оптимизация гиперпараметров

☒ Визуализация

Модель XGBClassifier

Датасет

Модель

Визуализация

Настройки блока : Модель XGBClassifier

Глубина дерева

2 50 2

Количество базовых моделей

2 2000 100

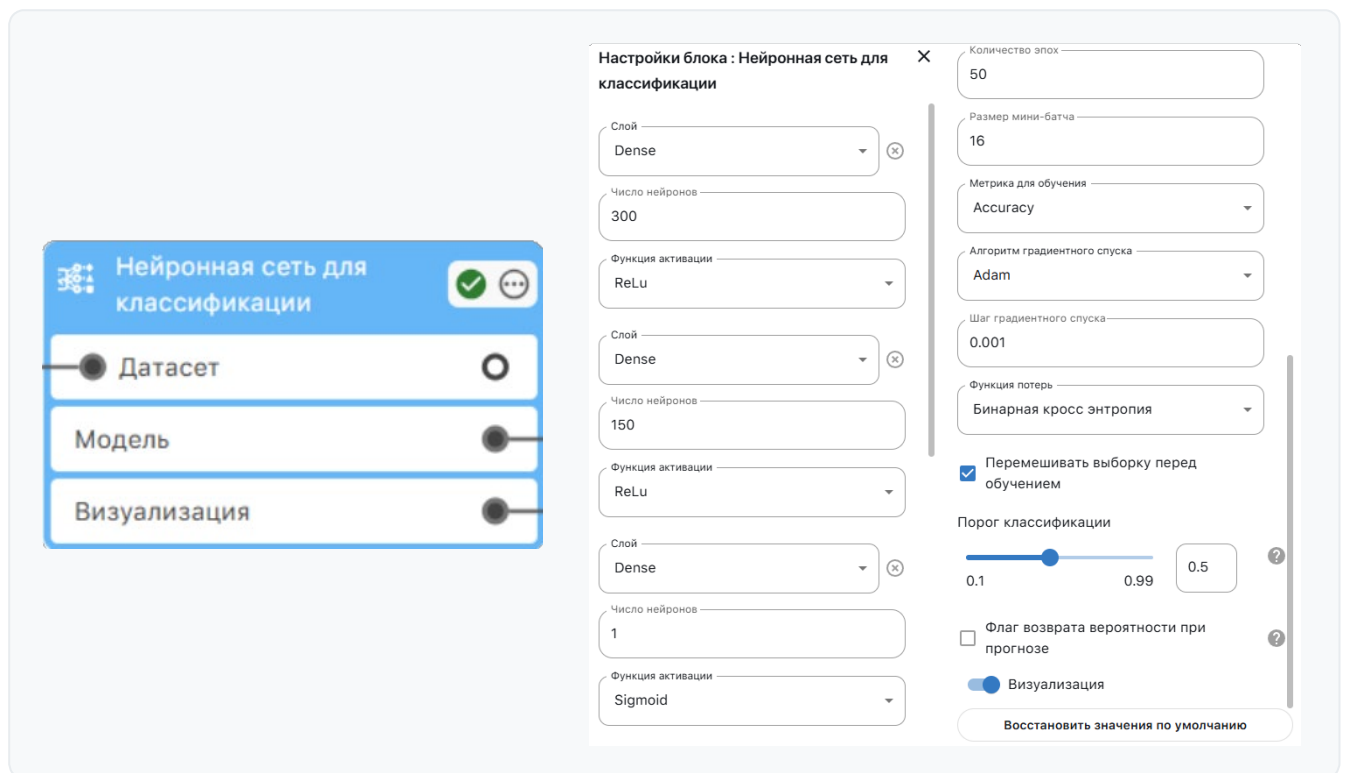
Порог классификации

0.1 0.9 0.5

☐ Флаг возврата вероятности при прогнозе

☐ Оптимизация гиперпараметров

☒ Визуализация



Анализ визуализаций обучения

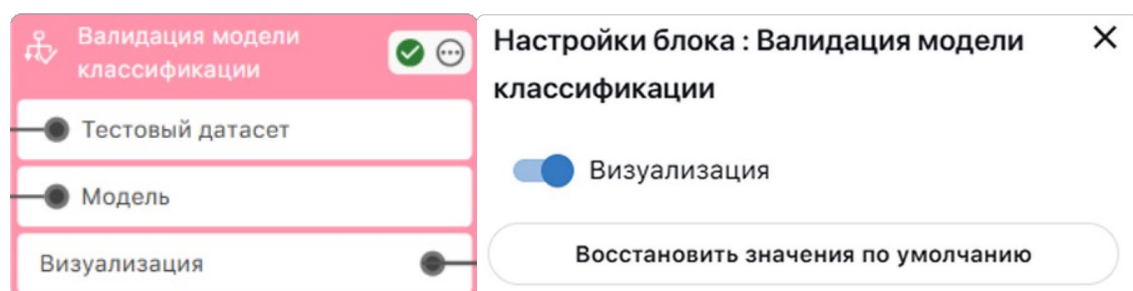
На графиках показано, как каждая из трех моделей разделила данные в трехмерном пространстве (созданном для наглядности с помощью метода главных компонент). Точки — это отдельные сообщения, а их цвет соответствует предсказанному классу.

Визуально все три модели смогли найти разделяющую плоскость между двумя классами, что говорит о том, что задача решается. Однако по этим графикам сложно судить, какая модель лучше. Для этого нужен следующий шаг — валидация.

Шаг 9. Валидация и сравнение моделей

На этом шаге одновременно проверим качество всех трех моделей с помощью блока

«Валидация модели классификации». Он получает на вход тестовую выборку и три обученные модели, после чего рассчитывает и отображает метрики качества для каждой из них, позволяя нам сделать обоснованный выбор.



50.2 Анализ результатов валидации

Блок валидации позволяет объективно оценить и сравнить производительность всех трех моделей на тестовых данных, которые не использовались при обучении. Анализ метрик помогает нам выбрать наиболее эффективный алгоритм для решения поставленной задачи.

1. Индивидуальные отчеты о классификации (Classification Reports)

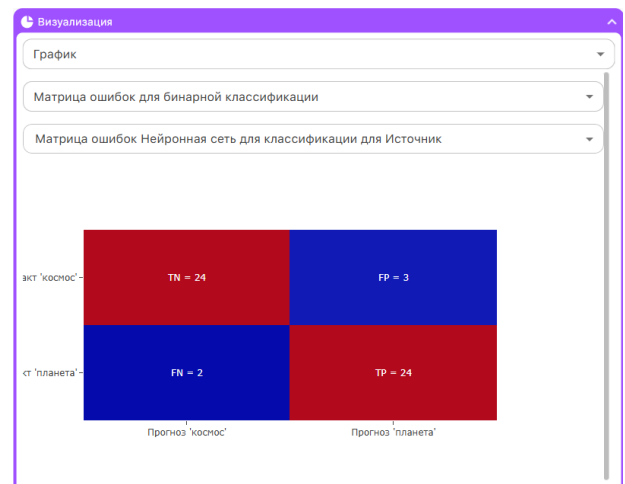
Для каждой модели формируется подробный отчет с метриками Precision (точность), Recall (полнота) и F1-score (гармоническое среднее). Мы можем проанализировать, как каждая модель справляется с отдельными классами («космос», «планета»). Для общей оценки моделей используется усредненное значение `macro avg F1-score`.

2. Сравнительная таблица и выбор лучшей модели

Сравнительная таблица агрегирует ключевые метрики, предоставляя итоговую картину для принятия решения. Основываясь на данных из визуализации, получаем следующие результаты:

- Нейронная сеть. F1-score = 0.906. Показывает наилучший результат по основной метрике.
- XGB классификатор. F1-score = 0.836. Результат ниже, чем у нейронной сети, но все еще находится на высоком уровне.
- К ближайших соседей (KNN). F1-score = 0.784. Демонстрирует самый низкий F1-score среди сравниваемых моделей.

Визуализация				
Таблица				
Сравнительная таблица				
	Accuracy	F1	Precision	Recall
Нейронная сеть для классификации_Источник	0.906	0.906	0.889	0.923
XGB классификатор(max_depth=2, n_estimators=100)_Источник	0.83	0.836	0.793	0.885
К ближайших соседей для классификации(n_neighbors=5, weights=uniform, metric=euclidean)_Источник	0.792	0.784	0.8	0.769



Вывод: сравнительный анализ метрик показывает, что для данной задачи классификации текстов наилучшую производительность продемонстрировала Нейронная сеть. Она достигла самых высоких значений как по F1-мере (0.91), так и по общей точности (Accuracy), что делает ее оптимальным выбором для дальнейшего использования.

50.3 Регрессия. Улучшение модели с помощью инженерии признаков

Задача

Предсказать числовой показатель — уровень послушания собаки — на основе её физических и категориальных характеристик.

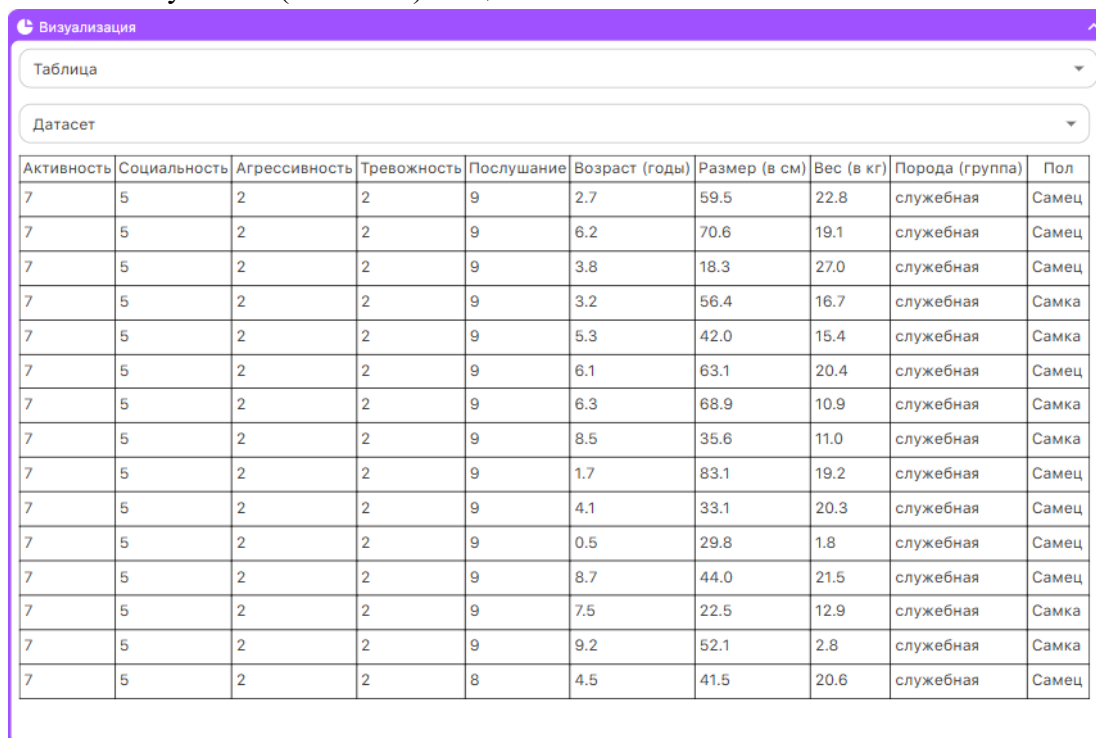
Цель

Таблица содержит три типа признаков, на основе которых мы будем строить предсказания:

- Поведенческие (числовые): Активность, Социальность, Агрессивность, Тревожность.
- Физические (числовые): Возраст (годы), Размер (в см), Вес (в кг).
- Категориальные: Порода (группа), Пол.

И целевая переменная, которую мы будем предсказывать:

- Послушание (числовая) — целевой показатель.



The screenshot shows a web application window titled "Визуализация" (Visualization). Inside, there are two dropdown menus: "Таблица" (Table) and "Датасет" (Dataset). Below these is a table with 10 columns: "Активность" (Activity), "Социальность" (Sociality), "Агрессивность" (Aggressiveness), "Тревожность" (Anxiety), "Послушание" (Obedience), "Возраст (годы)" (Age in years), "Размер (в см)" (Size in cm), "Вес (в кг)" (Weight in kg), "Порода (группа)" (Breed/Group), and "Пол" (Sex). The table contains 15 rows of data. The first four columns have values 7, 5, 2, and 2 respectively for all rows. The fifth column has values 9 for the first 14 rows and 8 for the last row. The remaining columns contain various numerical and categorical data points.

Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)	Порода (группа)	Пол
7	5	2	2	9	2.7	59.5	22.8	служебная	Самец
7	5	2	2	9	6.2	70.6	19.1	служебная	Самец
7	5	2	2	9	3.8	18.3	27.0	служебная	Самец
7	5	2	2	9	3.2	56.4	16.7	служебная	Самка
7	5	2	2	9	5.3	42.0	15.4	служебная	Самка
7	5	2	2	9	6.1	63.1	20.4	служебная	Самец
7	5	2	2	9	6.3	68.9	10.9	служебная	Самка
7	5	2	2	9	8.5	35.6	11.0	служебная	Самка
7	5	2	2	9	1.7	83.1	19.2	служебная	Самец
7	5	2	2	9	4.1	33.1	20.3	служебная	Самец
7	5	2	2	9	0.5	29.8	1.8	служебная	Самец
7	5	2	2	9	8.7	44.0	21.5	служебная	Самец
7	5	2	2	9	7.5	22.5	12.9	служебная	Самка
7	5	2	2	9	9.2	52.1	2.8	служебная	Самка
7	5	2	2	8	4.5	41.5	20.6	служебная	Самец

Шаг 2. Создание агрегированных признаков

Это начало ветки инженерии признаков. С помощью блока «Агрегация» группируем данные по категориальным признакам (Порода, Пол) и вычисляем средние значения числовых характеристик для каждой группы.

Настройка

В настройках блока выбираем статистику Среднее, указываем столбцы для группировки и столбцы для подсчёта статистики.

Агрегация
🔍 ⋮

Датасет
○

Визуализация
○

Настройки блока : Агрегация ✕

Статистика ?

Среднее
▼

Выгрузить признаки

Поведение собак
▼

Столбцы для группировки

Порода (группа) ✕
Пол ✕
▼
✕

Выгрузить признаки

Поведение собак
▼

Столбцы для подсчета статистики

Активность ✕

Социальность ✕

Агрессивность ✕

Тревожность ✕

Возраст (годы) ✕

Размер (в см) ✕

Вес (в кг) ✕

✕

Визуализация

Мы создаём новые признаки, которые описывают «контекст» для каждой собаки.

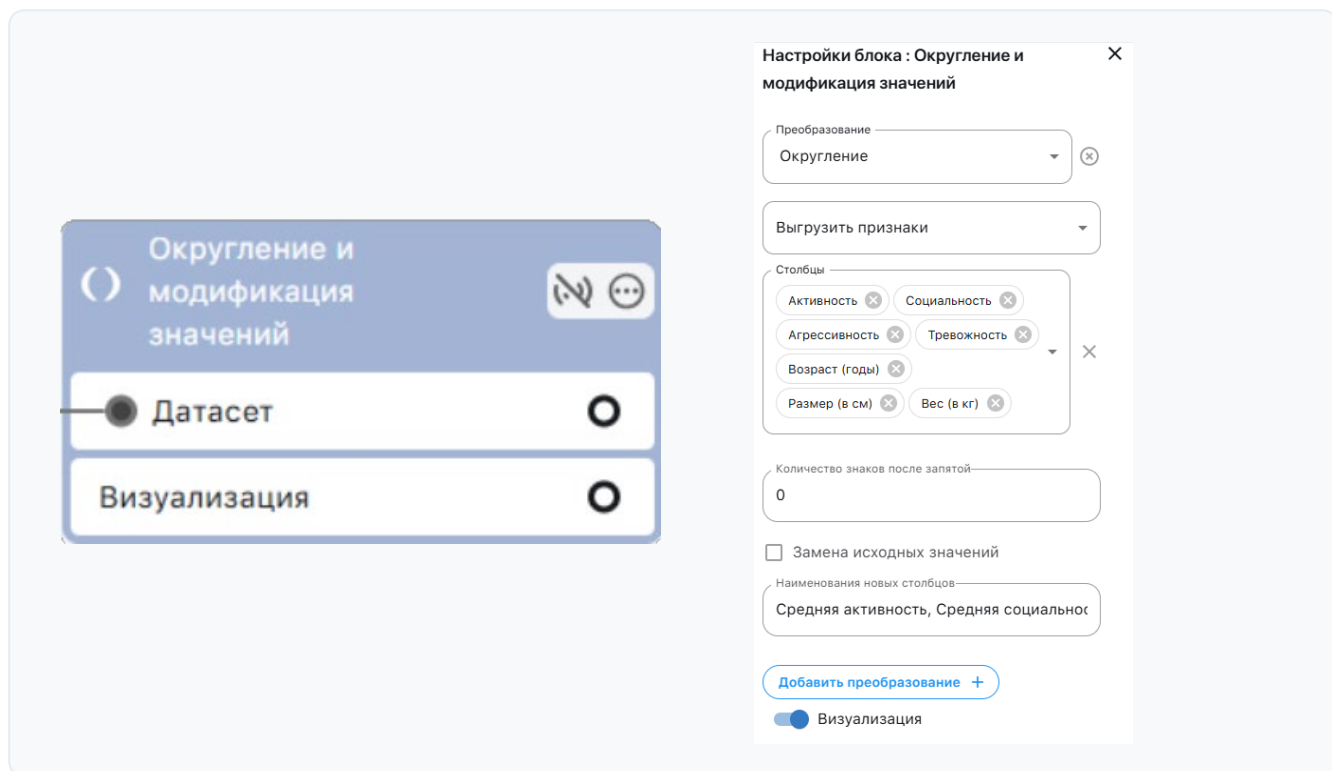
Визуализация								
Таблица								
Группировка и агрегирование								
Порода (группа)	Пол	Активность	Социальность	Агрессивность	Тревожность	Возраст (годы)	Размер (в см)	Вес (в кг)
декоративная	Самец	2.0	8.016666666666667	1.066666666666667	8.05	6.010000000000001	40.311666666666666	14.055
декоративная	Самка	1.96	8.08	1.04	8.0	5.774	41.828	14.72
охотничья	Самец	7.980392156862745	5.0	8.0	5.0	5.315686274509805	47.21176470588236	15.627450980392156
охотничья	Самка	7.979591836734694	5.0	8.0	5.0	5.324489795918367	45.28367346938776	15.277551020408163
пастушья	Самец	5.9743589743589745	4.0256410256410255	2.9743589743589745	3.9743589743589745	5.207692307692308	44.00512820512821	15.605128205128207
пастушья	Самка	5.902439024390244	3.951219512195122	3.0	4.024390243902439	5.0268292682926825	46.78536585365854	17.229268292682928
примитивная	Самец	4.891304347826087	5.021739130434782	5.021739130434782	4.891304347826087	4.856521739130435	43.55652173913043	16.430434782608696
примитивная	Самка	5.0454545454545454	4.954545454545454	5.0	5.0227272727272725	4.752272727272727	47.95227272727273	13.909090909090908
служебная	Самец	7.0	5.0	1.9846153846153847	2.0	5.2	47.215384615384615	15.930769230769231
служебная	Самка	7.0	5.0	2.0	2.018181818181818	5.236363636363636	46.70181818181818	13.969090909090909

Шаг 3. Округление значений

Для удобства и чистоты данных мы используем блок «Округление и модификация значений», чтобы округлить полученные средние значения.

Настройка

1. Выбираем Округление.
2. Указываем все столбцы со средними значениями, которые получили на предыдущем шаге.
3. Устанавливаем 0, чтобы получить целые числа.
4. Не включаем опцию «Замена исходных значений», чтобы блок создал новые столбцы.
5. Задаём им понятные имена.



В результате в датасете появляются новые столбцы с понятными именами и округлёнными до целых чисел значениями.

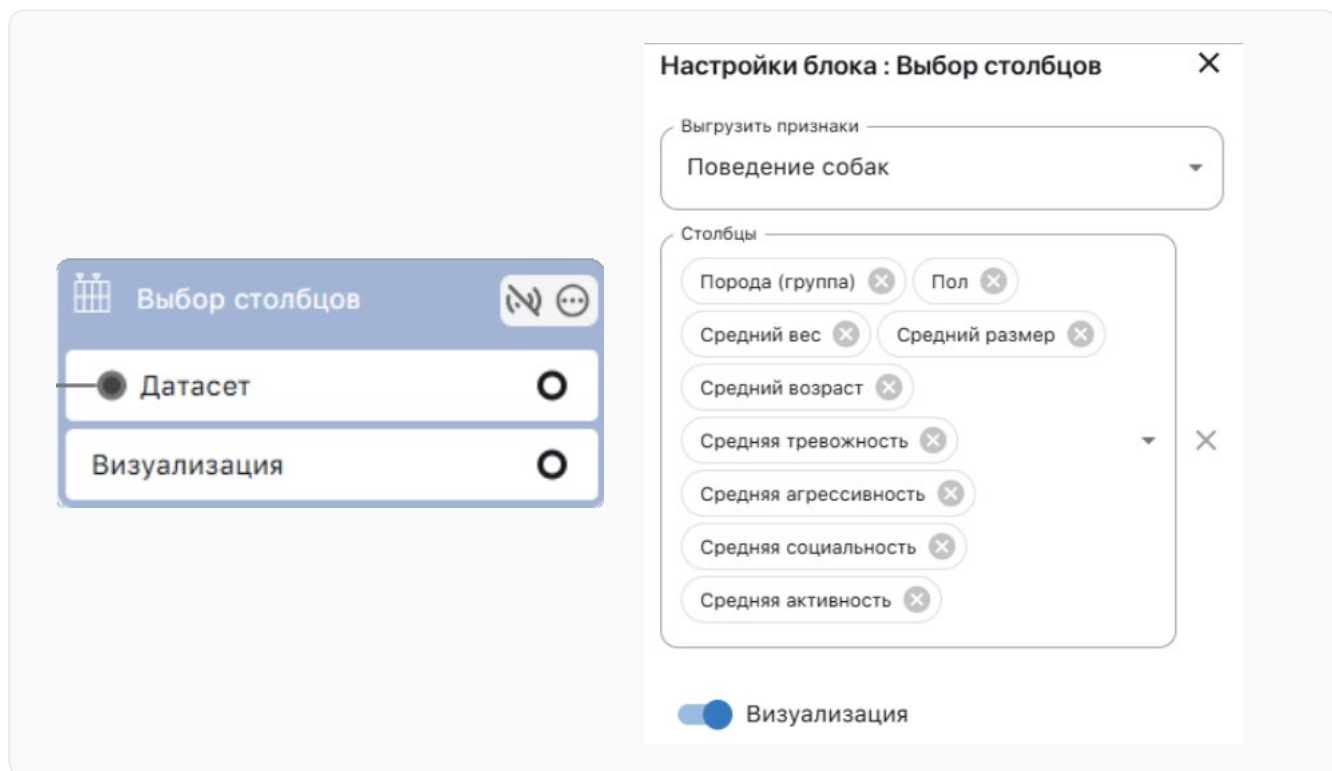
Порода (группа)	Пол	Активность	Социальность	Агрессивность	Тревожность	Возраст (годы)	Размер (в см)	Вес (в кг)	Средняя активность	Средняя социальность	Средняя агрессивность	Средняя тревожность	Средний возраст	Средний размер	Средний вес
декоративная	Самец	2.0	8.016666666666667	1.066666666666667	8.05	6.010000000000001	40.311666666666666	14.055	2.0	8.0	1.0	8.0	6.0	40.0	14.0
декоративная	Самка	1.96	8.08	1.04	8.0	5.774	41.828	14.72	2.0	8.0	1.0	8.0	6.0	42.0	15.0
охотничья	Самец	7.980392156862745	5.0	8.0	5.0	5.315686274509805	47.21176470588236	15.627450980392156	8.0	5.0	8.0	5.0	5.0	47.0	16.0
охотничья	Самка	7.979591836734694	5.0	8.0	5.0	5.324489795918367	45.28367346938776	15.277551020408163	8.0	5.0	8.0	5.0	5.0	45.0	15.0
пастушья	Самец	5.9743589743589745	4.0256410256410255	2.9743589743589745	3.9743589743589745	5.207692307692308	44.00512820512821	15.605128205128207	6.0	4.0	3.0	4.0	5.0	44.0	16.0
пастушья	Самка	5.902439024390244	3.951219512195122	3.0	4.024390243902439	5.0268292682926825	46.78536585365854	17.229268292682928	6.0	4.0	3.0	4.0	5.0	47.0	17.0
примитивная	Самец	4.891304347826087	5.021739130434782	5.021739130434782	4.891304347826087	4.856521739130435	43.55652173913043	16.430434782608696	5.0	5.0	5.0	5.0	5.0	44.0	16.0
примитивная	Самка	5.0454545454545454	4.954545454545454	5.0	5.0227272727272725	4.752272727272727	47.95227272727273	13.909090909090909	5.0	5.0	5.0	5.0	5.0	48.0	14.0
служебная	Самец	7.0	5.0	1.9846153846153847	2.0	5.2	47.215384615384615	15.930769230769231	7.0	5.0	2.0	2.0	5.0	47.0	16.0
служебная	Самка	7.0	5.0	2.0	2.018181818181818	5.236363636363636	46.70181818181818	13.969090909090909	7.0	5.0	2.0	2.0	5.0	47.0	14.0

Шаг 4. Выбор новых признаков

С помощью блока «Выбор столбцов» оставим только нужные столбцы, отбросив всё лишнее.

Настройка

Выбираем столбцы для группировки (Порода, Пол) и новые столбцы со средними значениями.



Окно визуализации демонстрирует результат работы блока — новую таблицу, которая содержит только выбранные столбцы.

Визуализация

Таблица

Измененный датасет после выборки

Порода (группа)	Пол	Средний вес	Средний размер	Средний возраст	Средняя тревожность	Средняя агрессивность	Средняя социальность	Средняя активность
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0
декоративная	Самка	15.0	42.0	6.0	8.0	1.0	8.0	2.0
охотничья	Самец	16.0	47.0	5.0	5.0	8.0	5.0	8.0
охотничья	Самка	15.0	45.0	5.0	5.0	8.0	5.0	8.0
пастушья	Самец	16.0	44.0	5.0	4.0	3.0	4.0	6.0
пастушья	Самка	17.0	47.0	5.0	4.0	3.0	4.0	6.0
примитивная	Самец	16.0	44.0	5.0	5.0	5.0	5.0	5.0
примитивная	Самка	14.0	48.0	5.0	5.0	5.0	5.0	5.0
служебная	Самец	16.0	47.0	5.0	2.0	2.0	5.0	7.0
служебная	Самка	14.0	47.0	5.0	2.0	2.0	5.0	7.0

Шаг 5. Обогащение датасета

С помощью блока «Объединение датасетов» присоединяем новые агрегированные признаки (из шага 4) к исходному датасету (из шага 1) по ключам Порода и Пол.

Настройка

1. В поле Тип соединения выбираем inner, чтобы в итоговую таблицу попали только те строки, для которых нашлись совпадения по ключам в обеих таблицах.
2. Указываем два датасета для объединения.
3. В качестве ключей для объединения и для левого, и для правого датафрейма указываем Порода (группа) и Пол.

Объединение датасетов
✓ ⋮

Левый датасет

Правый датасет

Итоговый датасет

Визуализация

Настройки блока : Объединение датасетов ✕

Тип соединения
▼

inner

Выгрузить признаки
▼

Поведение собак

Столбцы левого датасета для соединения
✕

Порода (группа) ✕

Пол ✕

▼
✕

Выгрузить признаки
▼

Поведение собак

Столбцы правого датасета для соединения
✕

Порода (группа) ✕

Пол ✕

▼
✕

Визуализация

Теперь для каждой собаки в датасете (каждой строки) доступны не только её индивидуальные признаки, но и новые, «контекстные» столбцы, созданные на предыдущих шагах:

- Средняя активность
- Средний возраст
- Средний вес и т.д.

Датасет готов к передаче в верхнюю, «продвинутую» ветку пайплайна для обучения модели.

Объединенная таблица																
Порода (группа)	Пол	Средний вес	Средний размер	Средний возраст	Средняя тревожность	Средняя агрессивность	Средняя социальность	Средняя активность	Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	9.6	30.4	7.9
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	3.1	29.0	-0.5
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	0.9	60.8	31.3
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	9	4	5.6	18.6	6.8
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	7	1	8	3	5.8	21.6	10.8
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	5.2	32.4	19.9
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	3	8	1	8	3	4.9	46.2	17.8
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	4.3	66.6	16.7
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	7	1	7	3	5.7	43.8	8.4
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	4.6	51.1	7.9
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	6.9	9.6	6.9
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	9	3	8.2	35.3	14.5
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	4	5.8	31.7	15.9
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	9	1	8	3	4.0	11.9	6.7
декоративная	Самец	14.0	40.0	6.0	8.0	1.0	8.0	2.0	2	8	1	8	3	7.7	30.8	6.0

Шаг 6. Параллельная подготовка данных

Теперь, когда у нас есть два источника данных (исходный и обогащённый), начинаем параллельную обработку. Следующие шаги выполняются одновременно для обеих веток.

В каждой ветке используем блок «Выбор признаков и целевых признаков», чтобы указать:

- Целевой признак: Послушание.
- Признаки: все остальные столбцы. В верхней («обогащённой») ветке этот набор будет включать и новые признаки.

The screenshot shows a workflow block titled 'Разделение датасета на обучающую и тестовую выборки' (Dataset splitting). The block has four ports: 'Датасет' (Dataset), 'Обучающий датасет' (Training dataset), 'Тестовый датасет' (Test dataset), and 'Визуализация' (Visualization). The 'Датасет' port is connected to a data source. The 'Обучающий датасет' and 'Тестовый датасет' ports are also connected. The 'Визуализация' port is not connected. To the right, the settings panel for this block is shown. It has a title 'Настройки блока : Выбор признаков и целевых признаков' (Block settings: Feature and target selection). There are two sections for 'Выгрузить признаки' (Export features), both set to 'Поведение собак' (Dog behavior). The first section, 'Признаки' (Features), lists 14 features: Активность, Социальность, Агрессивность, Тревожность, Возраст (годы), Размер (в см), Вес (в кг), Порода (группа), Пол, Средний вес, Средний размер, Средний возраст, Средняя тревожность, Средняя агрессивность, Средняя социальность, and Средняя активность. The second section, 'Целевые признаки' (Target features), lists 'Послушание' (Obedience).

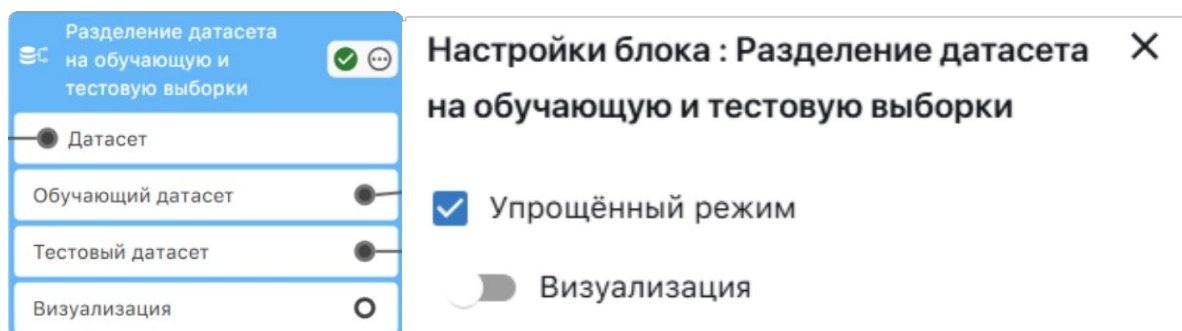
The screenshot shows a workflow block titled 'Выбор признаков и целевых признаков' (Feature and target selection). The block has two ports: 'Датасет' (Dataset) and 'Визуализация' (Visualization). The 'Датасет' port is connected to a data source. The 'Визуализация' port is not connected. To the right, the settings panel for this block is shown. It has a title 'Настройки блока : Выбор признаков и целевых признаков' (Block settings: Feature and target selection). There are two sections for 'Выгрузить признаки' (Export features), both set to 'Поведение собак' (Dog behavior). The first section, 'Признаки' (Features), lists 10 features: Активность, Социальность, Агрессивность, Тревожность, Возраст (годы), Размер (в см), Вес (в кг), Порода (группа), Пол, and Средний вес. The second section, 'Целевые признаки' (Target features), lists 'Послушание' (Obedience). At the bottom, there is a checkbox for 'Визуализация' (Visualization) which is currently unchecked.

Шаг 7. Разделение выборки

На этом этапе происходит одновременная подготовка данных в обеих ветках — базовой и обогащённой. Применяем блок «Разделение датасета на обучающую и тестовую выборки» к каждой ветке, чтобы разделить данные на обучающую и тестовую выборки.

Настройка

Оставляем «Упрощённый режим», который автоматически выполняет разделение в стандартной пропорции 80/20.

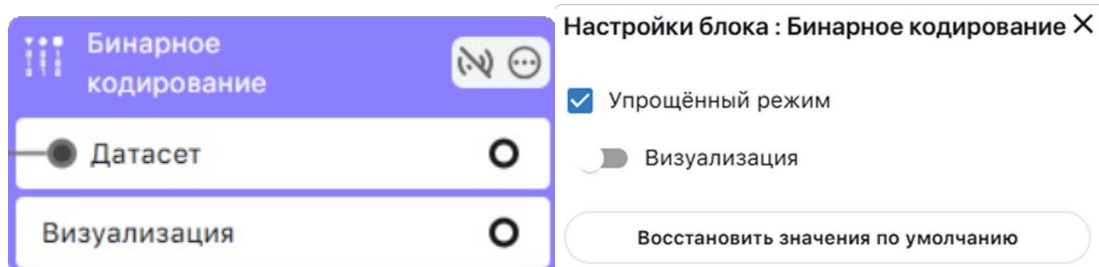


Шаг 8. Кодирование признаков

Категориальные признаки необходимо преобразовать в числовой формат. Для этой задачи в обеих ветках используем блок «Бинарное кодирование».

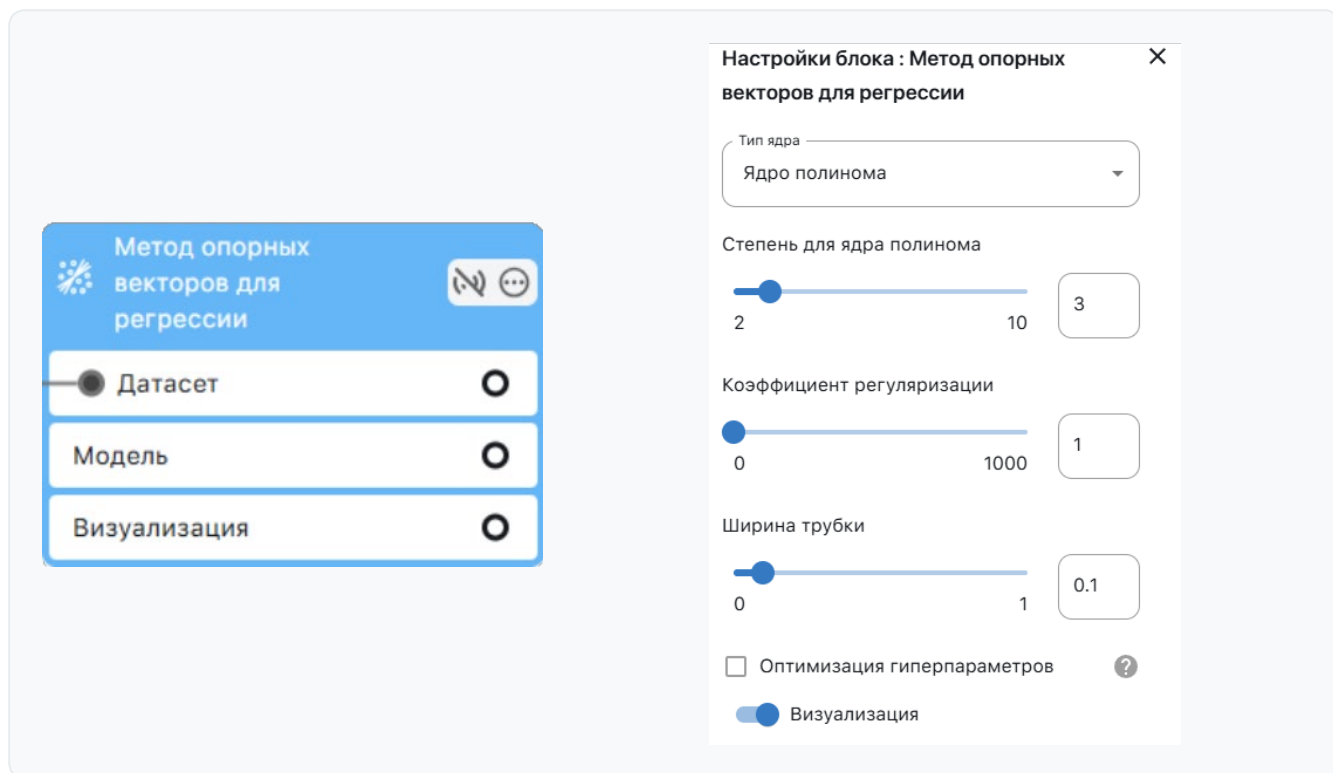
Настройка

Оставляем «Упрощённый режим», который автоматически находит и кодирует все категориальные признаки в датасете.



Шаг 9. Обучение моделей

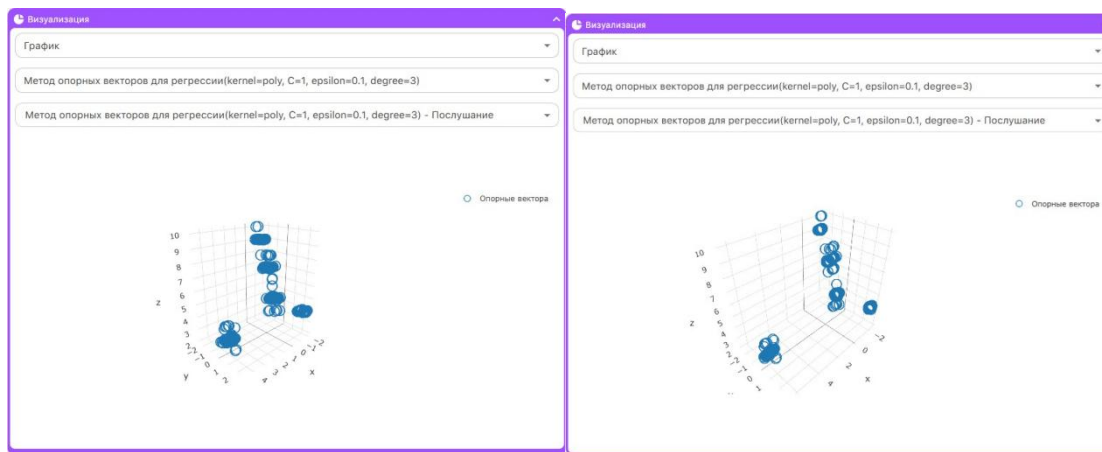
Добавляем два одинаковых блока «Метод опорных векторов для регрессии» — по одному в каждую из параллельных веток. Мы намеренно используем один и тот же алгоритм с одинаковыми настройками, чтобы единственным отличием между моделями были данные, на которых они обучались (исходные против обогащённых).



Анализ результатов обучения

Поскольку в датасете много признаков, для наглядности они сжимаются в три измерения с помощью метода главных компонент (PCA). Высота по оси Z соответствует уровню послушания. Точки на графике — опорные векторы.

- Левый график: модель на исходных данных. У модели есть только базовые признаки: возраст, вес, активность и т. д. Зависимость между ними и «Послушанием» может быть сложной и зашумлённой. Чтобы построить сложную разделяющую поверхность, модели приходится опираться на множество точек.
- Результат: большое количество опорных векторов. Это означает, что модель формирует извилистую поверхность, детально описывающую зависимость в данных, что создаёт риск переобучения.
 - Правый график: модель на обогащённых данных. Добавление новых, более информативных признаков упростило задачу для модели. Теперь ей не нужно учитывать множество индивидуальных особенностей каждой собаки — можно опираться на сильные, обобщающие признаки, и находить закономерности становится проще.
- Результат: меньшее количество опорных векторов. Это признак того, что модель выявила более устойчивую и обобщающую зависимость, опираясь на меньшее число ключевых точек.



Вывод: уменьшение числа опорных векторов — индикатор успешной инженерии признаков.

Как это повлияет на валидацию?

- Модель с большим количеством опорных векторов, скорее всего, хорошо покажет себя на обучающих данных, за счёт которых и строилась. Но на новых данных (валидационной выборке) есть высокий риск переобучения.
- Модель с меньшим количеством опорных векторов нашла более общую закономерность. Поэтому есть высокая вероятность, что она покажет себя лучше на валидационных данных. Её обобщающая способность выше.

Шаг 10. Валидация и сравнение моделей

На этом шаге мы одновременно проверяем качество обеих моделей с помощью блоков «Валидация модели регрессии». Каждый блок берёт тестовую выборку из своей ветви, сравнивает предсказания с реальными ответами и рассчитывает метрики качества.

Настройка

Блок не требует настроек. Просто подключите на вход тестовый датасет и обученную модель, затем в настройках блока активируйте визуализацию.

Валидация модели регрессии

Тестовый датасет

Модель

Визуализация

Настройки блока : Валидация модели регрессии

Визуализация

Восстановить значения по умолчанию

50.4 Анализ результатов валидации

Результаты валидации для обеих моделей представлены в виде четырёх блоков. Верхний ряд относится к модели с обогащёнными признаками, нижний — к базовой модели на исходных данных.

- Верхний ряд — результаты модели с новыми признаками (продвинутый подход).
- Нижний ряд — результаты базовой модели (на исходных данных).

1. Визуальное сравнение (графики «Предсказанные vs. Истинные»)

Эти графики показывают, насколько предсказания модели (ось Y) близки к реальным значениям (ось X). Идеальный результат — когда все точки лежат на диагонали.

- Базовая модель (нижний график): точки на графике сильно разбросаны, что говорит о значительных ошибках в предсказаниях.
- Улучшенная модель (верхний график): здесь точки сгруппированы гораздо плотнее вокруг диагонали. Визуально очевидно, что эта модель точнее.

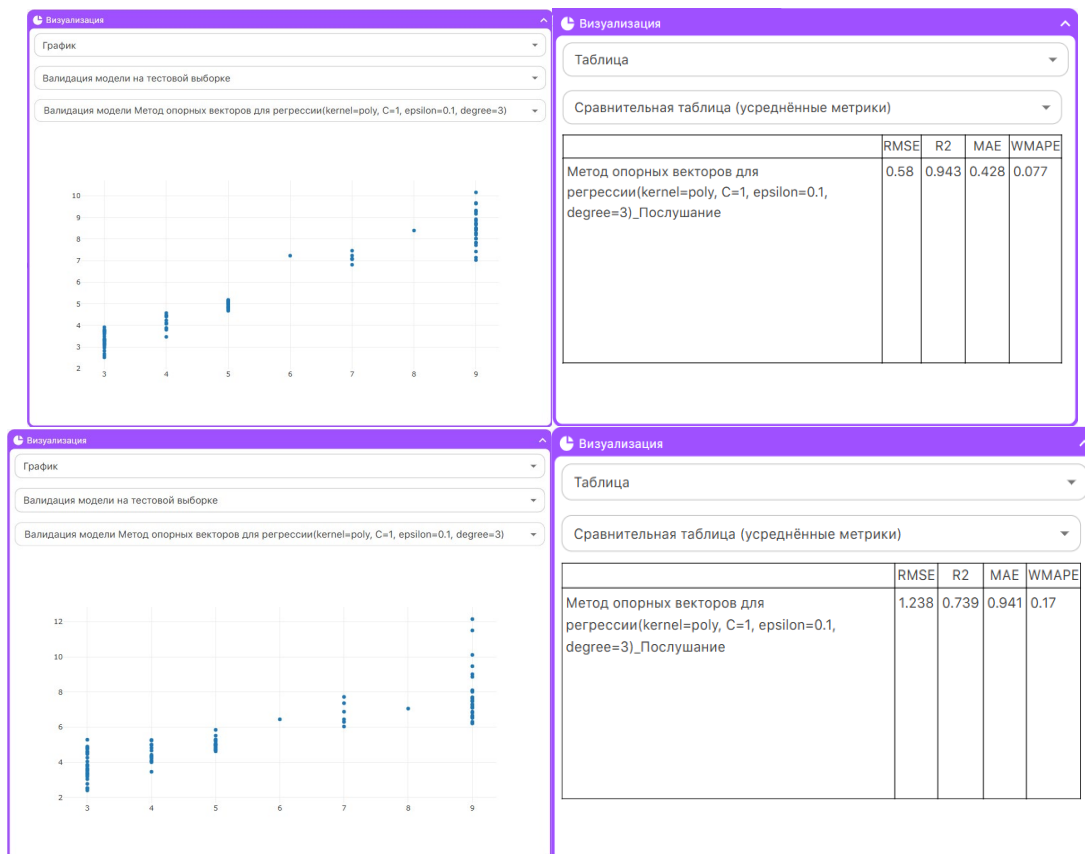
2. Сравнение по метрикам (таблицы)

Точную оценку дают числовые метрики, в первую очередь — R^2 (коэффициент детерминации).

- Базовая модель (нижняя ветвь): $R^2 = 0.58$. Модель на исходных данных смогла объяснить только 58% изменчивости в уровне послушания.
- Модель с новыми признаками (верхняя ветвь): $R^2 = 0.739$. Эта модель объясняет уже почти 74% изменчивости.

Остальные метрики показывают, насколько сильно модель ошибается в среднем.

- Средняя абсолютная ошибка (MAE): у базовой модели ошибка составляет 0.941, а у улучшенной — всего 0.428. В среднем предсказания модели с новыми признаками более чем в два раза точнее.
- Средняя ошибка в процентах (WMAPE): базовая модель ошибается в среднем на 17%, в то время как улучшенная — всего на 7.7%. Относительная ошибка также сократилась более чем вдвое.



50.5 Кластеризация. Поиск групп с помощью агрегированных признаков

Задача

Провести кластеризацию животных на основе их физических и поведенческих характеристик. Сравнить результаты кластеризации на исходных данных и на данных, обогащенных новыми признаками, созданными с помощью агрегации.

Цель

Продemonстрировать, как инженерия признаков (feature engineering) может улучшить качество кластеризации. Показать, что добавление признаков, описывающих средние характеристики групп, помогает алгоритму находить более осмысленную и интерпретируемую структуру в данных.

Датасет

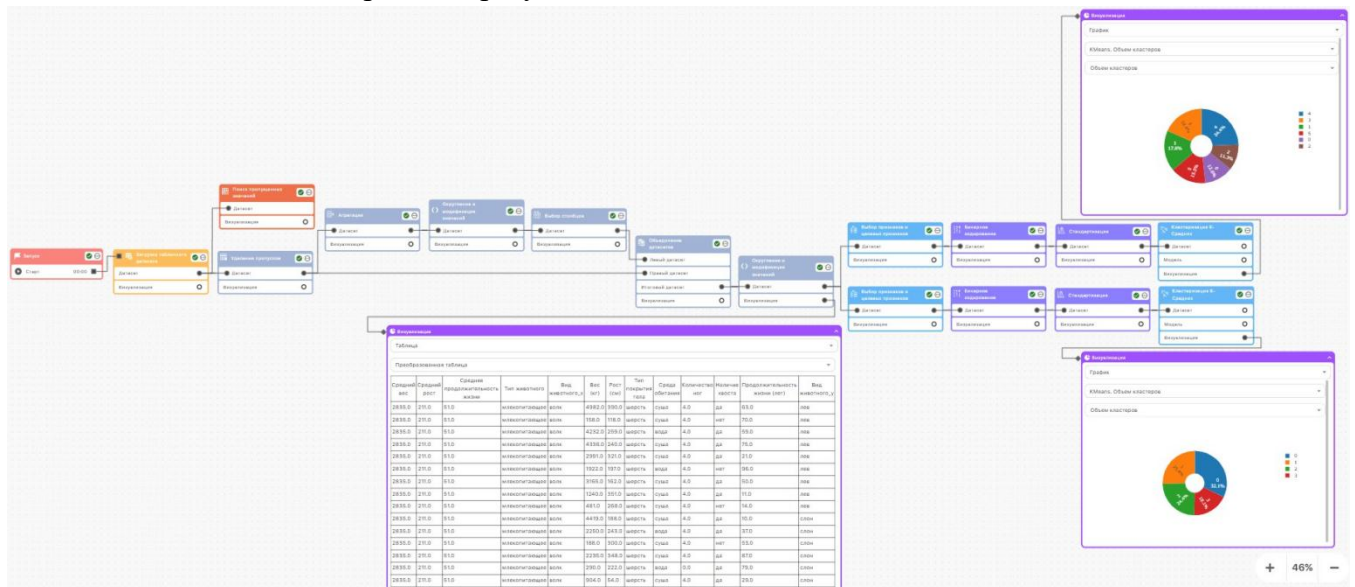
Будет использоваться датасет «Животные виды и типы», который содержит следующие столбцы:

- Вес (кг)
- Рост (см)
- Тип покрытия тела (категориальный)
- Среда обитания (категориальный)

- Количество ног
- Наличие хвоста (категориальный)
- Продолжительность жизни (лет)
- Тип животного (категориальный)
- Вид животного (категориальный)

Построение пайплайна

Для решения задачи будет построен пайплайн с двумя параллельными ветками обработки данных, чтобы наглядно сравнить результаты.



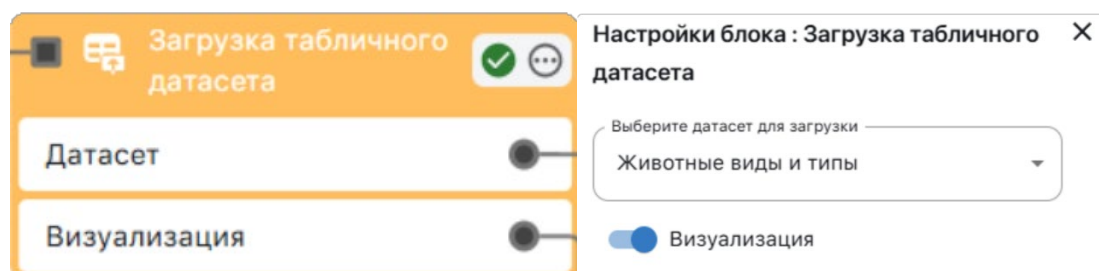
Шаг 1. Загрузка данных

Загрузим данные, используя блок «Загрузка табличного датасета». Этот датасет станет основой для всего эксперимента.

Настройка

В настройках блока в выпадающем списке указываем датасет «Животные виды и типы».

Активируем визуализацию, чтобы убедиться, что данные загрузились корректно.



В исходной таблице присутствуют числовые и категориальные признаки, а также пропущенные значения.

Визуализация								
Таблица								
Датасет								
Вес (kg)	Рост (см)	Тип покрытия тела	Среда обитания	Количество ног	Наличие хвоста	Продолжительность жизни (лет)	Тип животного	Вид животного
5.680647770825701	143.10000289738826	перья	суша	2	да	39	птица	орел
6.742908017518507	23.996488214520404	перья	воздух	2	да	40	птица	орел
10.650281409161078	12.881829201412344	перья	вода	2	нет	38	птица	орел
0.1116036110311134	148.90961830077046	перья	воздух	2	да	44	птица	орел
0.4436301331170947	83.46845243617449	перья	воздух	2	да	16	птица	орел
3.568292972411533	22.684900834594917	перья	воздух	2	да	3	птица	орел
12.913112060371176	105.24305540228916	перья	воздух	2	да	25	птица	орел
14.487917292810934	123.17562873630456	перья	воздух	2	да	20	птица	орел
3.691279443787575	105.65689263556416	перья	воздух	2	нет	35	птица	орел
13.648873990973852	46.229197424002365	перья	воздух	2	да	6	птица	орел
3.1983307767360136	89.47804589479881	перья	воздух	2	нет	44	птица	сокол
14.098534229306418	135.27582905987083	перья	воздух	2	нет	8	птица	сокол
8.599615218640452	82.91679640361531	перья	вода	2	нет	45	птица	сокол
<NA>	126.0232512812701	перья	воздух	2	да	45	птица	сокол
2.199770952123964	122.30757730556556	перья	воздух	2	нет	24	птица	сокол

Шаг 2. Поиск пропущенных значений

Перед любой обработкой необходимо оценить качество данных. Используем блок «Поиск пропущенных значений», чтобы определить, в каких столбцах и в каком количестве отсутствуют данные.

Настройка

Оставляем настройки по умолчанию, активировав «Упрощенный режим».

Поиск пропущенных значений

Датасет

Визуализация

Настройки блока : Поиск пропущенных значений

☒ Упрощенный режим

☒ Визуализация

Анализ визуализации показывает, что пропуски есть почти во всех столбцах.

Визуализация

Таблица

Количество пропусков

	amount_nan	percent_nan
Вес (кг)	2	0.50%
Рост (см)	3	0.75%
Тип покрытия тела	2	0.50%
Среда обитания	4	1.00%
Количество ног	1	0.25%
Наличие хвоста	4	1.00%
Продолжительность жизни (лет)	4	1.00%
Тип животного	0	0.00%
Вид животного	0	0.00%

Шаг 3. Удаление пропусков

Самый простой способ борьбы с пропусками — удаление строк, в которых они содержатся. Добавим на рабочую область блок «Удаление пропусков».

Настройка

Используем «Упрощенный режим». Он автоматически удалит все строки, содержащие хотя бы одно пропущенное значение.

Удаление пропусков

Датасет

Визуализация

Настройки блока : Удаление пропусков

☒ Упрощённый режим

☒ Визуализация

В окне визуализации видно, что пропущенные значения удалены.

Визуализация								
Таблица								
Датасет после удаления пропусков								
Вес (кг)	Рост (см)	Тип покрытия тела	Среда обитания	Количество ног	Наличие хвоста	Продолжительность жизни (лет)	Тип животного	Вид животного
5.680647770825701	143.10000289738826	перья	суша	2.0	да	39.0	птица	орел
6.742908017518507	23.996488214520404	перья	воздух	2.0	да	40.0	птица	орел
10.650281409161078	12.881829201412344	перья	вода	2.0	нет	38.0	птица	орел
0.1116036110311134	148.90961830077046	перья	воздух	2.0	да	44.0	птица	орел
0.4436301331170947	83.46845243617449	перья	воздух	2.0	да	16.0	птица	орел
3.568292972411533	22.684900834594917	перья	воздух	2.0	да	3.0	птица	орел
12.913112060371176	105.24305540228916	перья	воздух	2.0	да	25.0	птица	орел
14.487917292810934	123.17562873630456	перья	воздух	2.0	да	20.0	птица	орел
3.691279443787575	105.65689263556416	перья	воздух	2.0	нет	35.0	птица	орел
13.648873990973852	46.229197424002365	перья	воздух	2.0	да	6.0	птица	орел
3.1983307767360136	89.47804589479881	перья	воздух	2.0	нет	44.0	птица	сокол
14.098534229306418	135.27582905987083	перья	воздух	2.0	нет	8.0	птица	сокол
8.599615218640452	82.91679640361531	перья	вода	2.0	нет	45.0	птица	сокол
2.199770952123964	122.30757730556556	перья	воздух	2.0	нет	24.0	птица	сокол
5.983734620815988	51.088344460525335	перья	воздух	2.0	да	8.0	птица	сокол

Шаг 4. Агрегация

С помощью блока «Агрегация» сгруппируем данные по категориальным признакам (Тип животного, Вид животного) и вычислим средние значения числовых характеристик для каждой группы.

Настройка

1. Указываем, что хотим вычислить Среднее значение.
2. Указываем столбцы для группировки: Тип животного и Вид животного.
3. Выбираем столбцы для подсчета статистики: Вес (кг), Рост (см) и Продолжительность жизни (лет).

Агрегация

Датасет
Визуализация

Настройки блока : Агрегация

Статистика

Среднее

Выгрузить признаки

Животные виды и типы

Столбцы для группировки

Тип животного

Вид животного

Выгрузить признаки

Животные виды и типы

Столбцы для подсчета статистики

Вес (кг)

Рост (см)

Продолжительность жизни (лет)

Визуализация

В результате получаем новую таблицу, где для каждой комбинации типа и вида животного рассчитаны средние характеристики.

Тип животного	Вид животного	Вес (кг)	Рост (см)	Продолжительность жизни (лет)
млекопитающее	волк	2835.297327293876	210.50600773352775	51.0
млекопитающее	дельфин	2334.5159356405497	248.05168097952347	55.1
млекопитающее	жираф	3013.052912038124	172.37846151719404	48.666666666666664
млекопитающее	зебра	2301.469731516419	177.38176153083145	39.333333333333336
млекопитающее	кит	2939.981803178285	235.86684019487114	51.6
млекопитающее	лев	2611.7612067070577	256.2082939956175	51.0
млекопитающее	медведь	3124.9024625483335	229.5691024950973	54.0
млекопитающее	носорог	2652.6374797960807	224.64775873384352	49.6
млекопитающее	слон	1875.0768163344346	227.10484066827297	53.0
млекопитающее	тигр	3057.891562660683	226.71520549128832	52.4
птица	аист	9.098512435272186	61.36326268931012	29.444444444444443
птица	воробей	6.068432828837903	78.87797706121498	21.75
птица	голубь	7.505975955936881	92.45982117827272	20.9
птица	дятел	7.620789901738201	62.37045517414724	27.0
птица	орел	7.193854670200857	81.53460660830211	26.6

Шаг 5. Округление и модификация значений

Результаты агрегации содержат много знаков после запятой. Для удобства и последующего переименования столбцов используем блок «Округление и модификация значений».

Настройка

1. В качестве преобразования указываем Округление.
2. Выбираем все столбцы со средними значениями (Вес (кг), Рост (см), Продолжительность жизни (лет)).
3. Указываем количество знаков после запятой равным 0.
4. Активируем чек-бокс Замена исходных значений:.
5. Задаем новые имена: Средний вес, Средний рост, Средняя продолжительность жизни.

Округление и модификация значений

Датасет

Визуализация

Настройки блока : Округление и модификация значений

Преобразование

Округление

Выгрузить признаки

Животные виды и типы

Столбцы

Вес (кг)

Рост (см)

Продолжительность жизни (лет)

Количество знаков после запятой

0

Замена исходных значений

Наименования новых столбцов

Средний вес, Средний рост, Средняя про

Добавить преобразование +

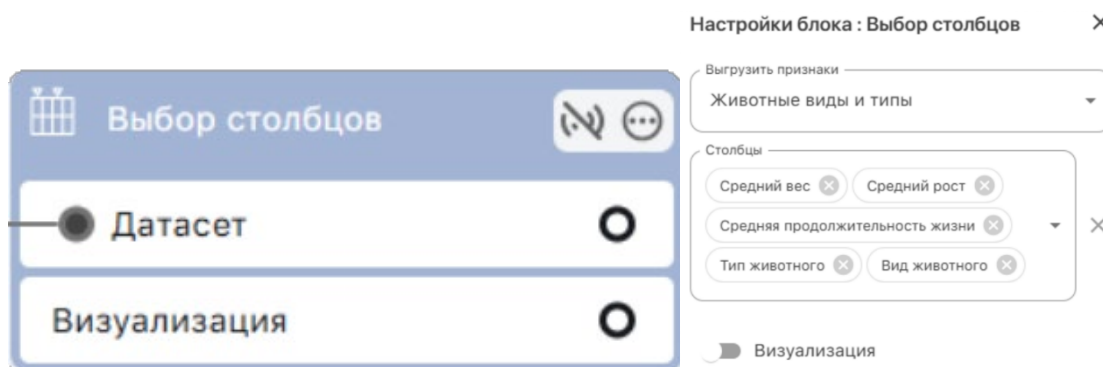
Визуализация

Шаг 6. Выбор столбцов

Из агрегированной таблицы нужны только новые признаки и ключи для объединения. Отбросим лишнее с помощью блока «Выбор столбцов».

Настройка

Выбираем столбцы для группировки (Тип животного, Вид животного) и новые столбцы со средними значениями, которые были созданы и переименованы на предыдущих шагах.

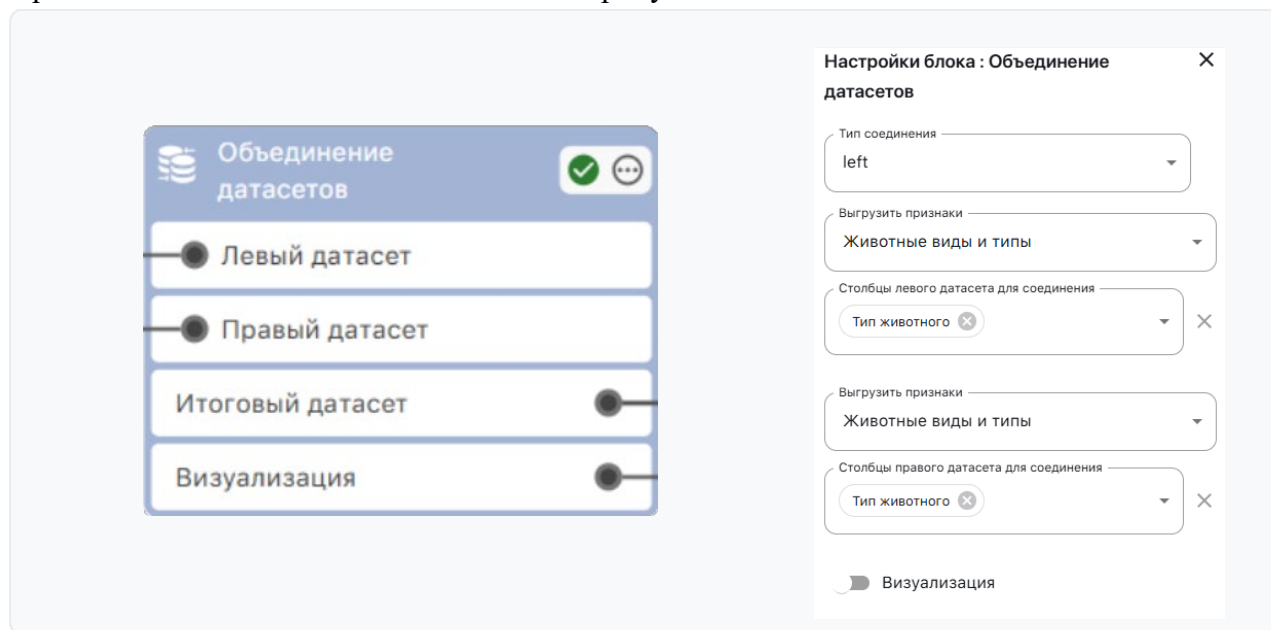


Шаг 7. Объединение датасетов

Теперь обогатим исходный датасет новыми признаками. Соединяем датасет с таблицей агрегированных признаков. Используем блок «Объединение датасетов», чтобы добавить созданные средние значения к каждой строке исходного датасета.

Настройка

1. В качестве типа соединения указываем left. Это сохранит все строки из исходного датасета.
2. Указываем столбцы левого/правого датасета для соединения: Тип животного. Объединяем данные на основе типа животного, чтобы каждая запись получила информацию о средних показателях по своему типу.
3. Левый датасет - выход блока "Выбор столбцов".
4. Правый датасет - выход блока "Удаление пропусков".



Шаг 8. Округление и модификация значений

Для единообразия и наглядности округлим числовые столбцы Вес (кг) и Рост (см) в только что объединенном датасете, с помощью блока «Округление и модификация значений».

Настройка

Выполняем операцию Округления до 0 знаков после запятой для столбцов Вес (кг) и Рост (см) с заменой исходных значений.

К дальнейшей обработке подготовлено два датасета: один исходный (выход блока "Удаление пропусков") и второй обогащенный средними значениями по группам (выход данного блока).

Визуализация													
Таблица													
Преобразованная таблица													
Средний вес	Средний рост	Средняя продолжительность жизни	Тип животного	Вид животного_x	Вес (кг)	Рост (см)	Тип покрытия тела	Среда обитания	Количество ног	Наличие хвоста	Продолжительность жизни (лет)	Вид животного_y	
2835.0	211.0	51.0	млекопитающее	волк	4982.0	390.0	шерсть	суша	4.0	да	63.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	158.0	118.0	шерсть	суша	4.0	нет	70.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	4232.0	259.0	шерсть	вода	4.0	да	59.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	4336.0	240.0	шерсть	суша	4.0	да	75.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	2991.0	321.0	шерсть	суша	4.0	да	21.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	1922.0	197.0	шерсть	вода	4.0	нет	96.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	3165.0	162.0	шерсть	суша	4.0	да	50.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	1240.0	351.0	шерсть	суша	4.0	да	11.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	481.0	268.0	шерсть	суша	4.0	нет	14.0	лев	
2835.0	211.0	51.0	млекопитающее	волк	4419.0	188.0	шерсть	суша	4.0	да	10.0	слон	
2835.0	211.0	51.0	млекопитающее	волк	2250.0	243.0	шерсть	вода	4.0	да	37.0	слон	
2835.0	211.0	51.0	млекопитающее	волк	188.0	300.0	шерсть	суша	4.0	нет	53.0	слон	
2835.0	211.0	51.0	млекопитающее	волк	2235.0	348.0	шерсть	суша	4.0	да	87.0	слон	
2835.0	211.0	51.0	млекопитающее	волк	290.0	222.0	шерсть	вода	0.0	да	79.0	слон	
2835.0	211.0	51.0	млекопитающее	волк	904.0	54.0	шерсть	суша	4.0	да	29.0	слон	

Шаг 9. Параллельная подготовка данных

Теперь пайплайн разветвляется. Будем готовить данные для двух моделей параллельно, чтобы сравнить результаты. В обеих ветках используем блок «Выбор признаков и целевых признаков».

- Верхняя ветка (исходные признаки). В поле «Признаки» выбираем все исходные столбцы, кроме Тип животного и Вид животного. Исключаем их, так как это неявные ответы, которые алгоритм должен найти.
- Нижняя ветка (добавлены новые признаки). Здесь также выбираем исходные признаки, но добавляем новые, созданные на этапе агрегации: Средний вес, Средний рост и Средняя продолжительность жизни.

The image displays two instances of the 'Feature and Target Selection' block, each with its configuration panel open. The block interface includes a title bar with a checkmark and a menu icon, and two main sections: 'Dataset' and 'Visualization'.

Top Configuration (Initial Features):

- Dataset:** Selected.
- Visualization:** Not selected.
- Settings Panel:**
 - Export features:** 'Animal types and types'.
 - Features:** 'Вес (кг)', 'Рост (см)', 'Тип покрытия тела', 'Среда обитания', 'Количество ног', 'Наличие хвоста', 'Продолжительность жизни (лет)'.
 - Export features:** (empty dropdown).
 - Target features:** (empty dropdown).
 - Visualization:** Disabled.

Bottom Configuration (Features with Aggregations):

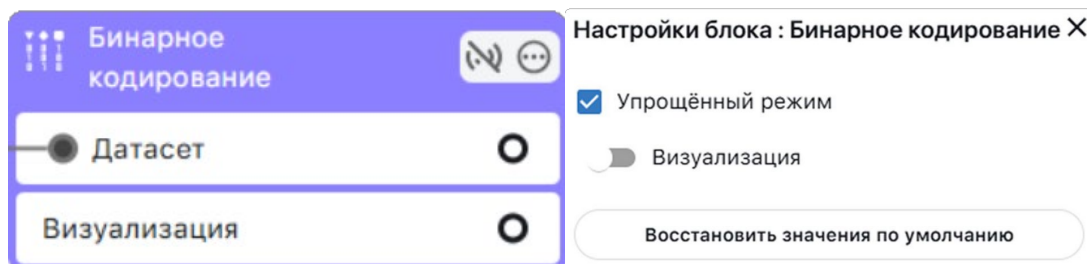
- Dataset:** Selected.
- Visualization:** Not selected.
- Settings Panel:**
 - Export features:** 'Animal types and types'.
 - Features:** 'Тип покрытия тела', 'Среда обитания', 'Количество ног', 'Наличие хвоста', 'Средний вес', 'Средний рост', 'Средняя продолжительность жизни'.
 - Export features:** (empty dropdown).
 - Target features:** (empty dropdown).
 - Visualization:** Disabled.

Шаг 10. Кодирование признаков

Алгоритмы кластеризации работают с числами, поэтому категориальные признаки необходимо преобразовать в числовой формат. Используем блок «Бинарное кодирование» в обеих ветках для решения этой задачи.

Настройка

Оставляем «Упрощённый режим», который автоматически находит и кодирует все категориальные признаки в датасете.

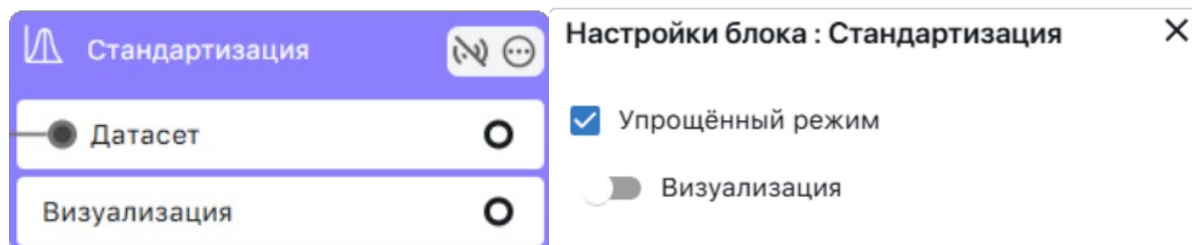


Шаг 11. Стандартизация

Признаки имеют разный масштаб (вес в кг, рост в см). Чтобы алгоритм K-Means не отдавал необоснованное предпочтение признакам с большими значениями, необходимо привести их к одному масштабу с помощью стандартизации.

Настройка

В обеих ветках используем блок «Стандартизация» в «Упрощённом режиме». Он автоматически обработает все числовые признаки.



Шаг 12. Обучение моделей и анализ результатов

Финальный этап — применение алгоритма «Кластеризация К-Средних» к подготовленным данным в обеих ветках.

Настройка

В обоих блоках включаем «Автоматический подбор количества кластеров» с использованием «Метода локтя». Это позволит платформе самой определить оптимальное K (количество кластеров) для каждого из наборов данных. Активируем визуализацию для анализа.

Кластеризация К-Средних

Датасет

Модель

Визуализация

Настройки блока : Кластеризация К-Средних

☐ Упрощённый режим

☒ Автоматический подбор количества кластеров

Метод подбора количества кластеров

Метод локтя

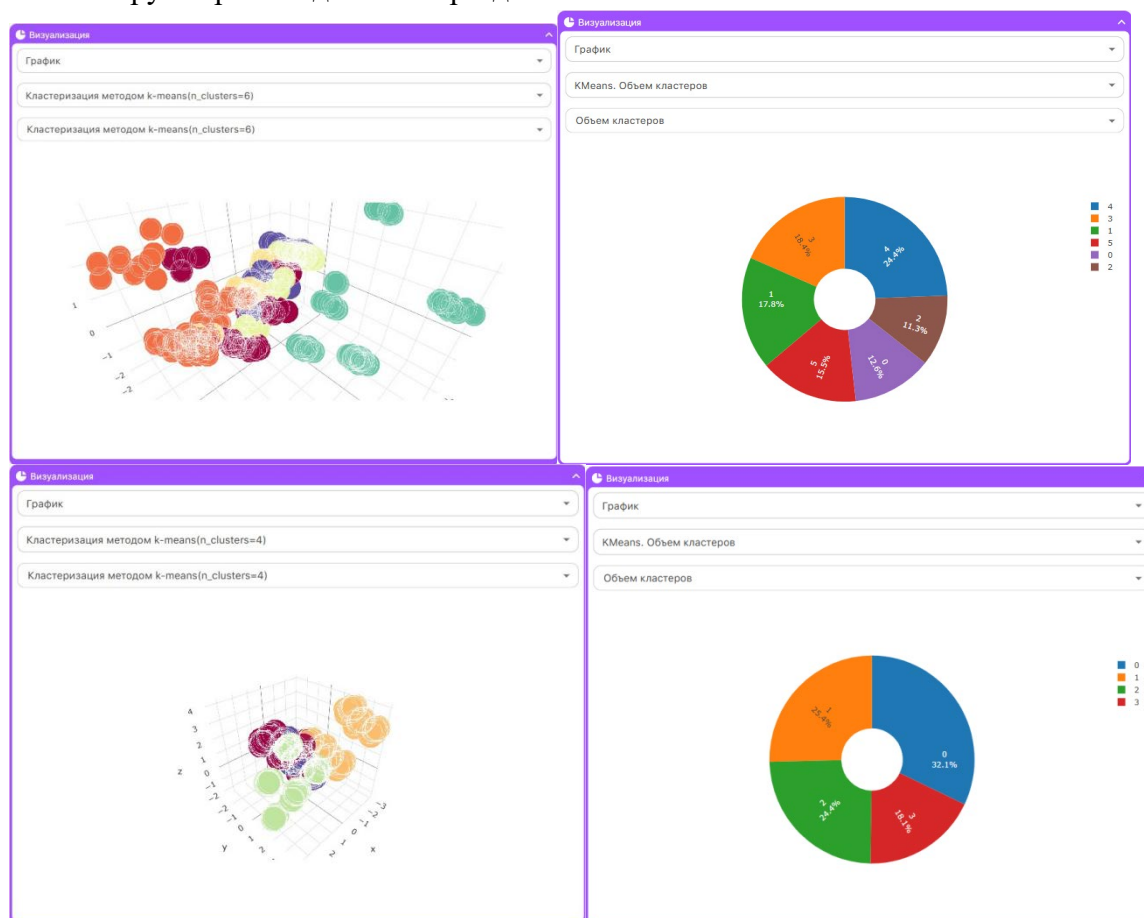
☐ Оптимизация гиперпараметров

☒ Визуализация

Анализ результатов

Визуализации наглядно демонстрируют разницу между двумя подходами и подтверждают гипотезу:

- Модель без новых признаков (верхние графики). Алгоритм определил оптимальное количество кластеров равное 6. Это не соответствует ожидаемой биологической классификации (птицы, рыбы, рептилии, млекопитающие). Модель нашла математические группы, но их сложно интерпретировать.
- Модель с агрегированными признаками (нижние графики). Алгоритм нашел 4 кластера. Это число точно совпадает с количеством уникальных типов животных в исходном датасете. Новые признаки дали модели подсказку, позволив ей сгруппировать данные гораздо более осмысленно.



Вывод: новые, агрегированные по типу, признаки улучшили качество кластеризации. Они помогли алгоритму "увидеть" скрытую структуру данных и выделить группы, которые соответствуют реальным биологическим типам.

Важное замечание об "утечке данных" (Data Leakage):

На этапе агрегации мы использовали столбцы Тип животного и Вид животного для расчета средних. По сути, мы использовали "ответ" (принадлежность к типу), чтобы создать признаки, которые затем помогли этот "ответ" найти. В реальной задаче, где принадлежность к классу неизвестна, это было бы методологической ошибкой. Однако в данном учебном примере это является отличной демонстрацией того, насколько сильно качественные, контекстные признаки влияют на результат кластерного анализа.

51. Использование модели ИИ

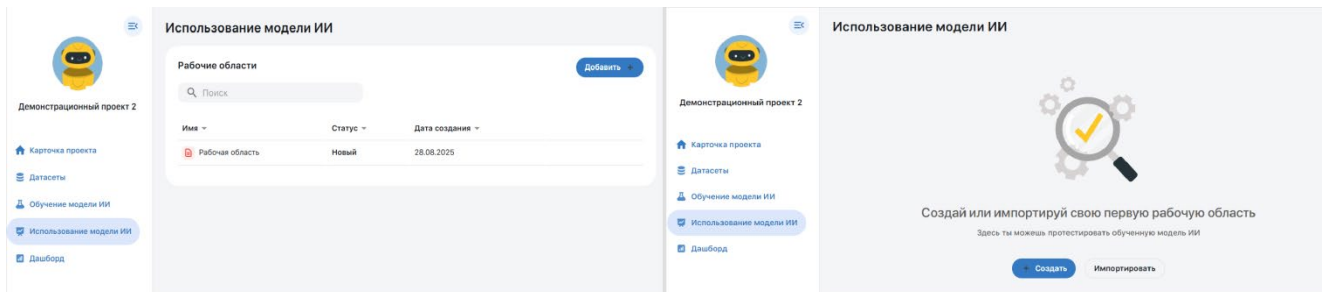
После того как модель успешно обучена и сохранена, наступает этап её применения для решения задач. Этот процесс, известный как инференс (inference), заключается в использовании готовой модели для получения прогнозов на новых данных, для которых ответ заранее неизвестен.

В разделе «Использование модели ИИ» мы также собираем конструктор из блоков.

В следующем разделе мы рассмотрим практические примеры построения таких пайплайнов.

51.1 Управление рабочими областями

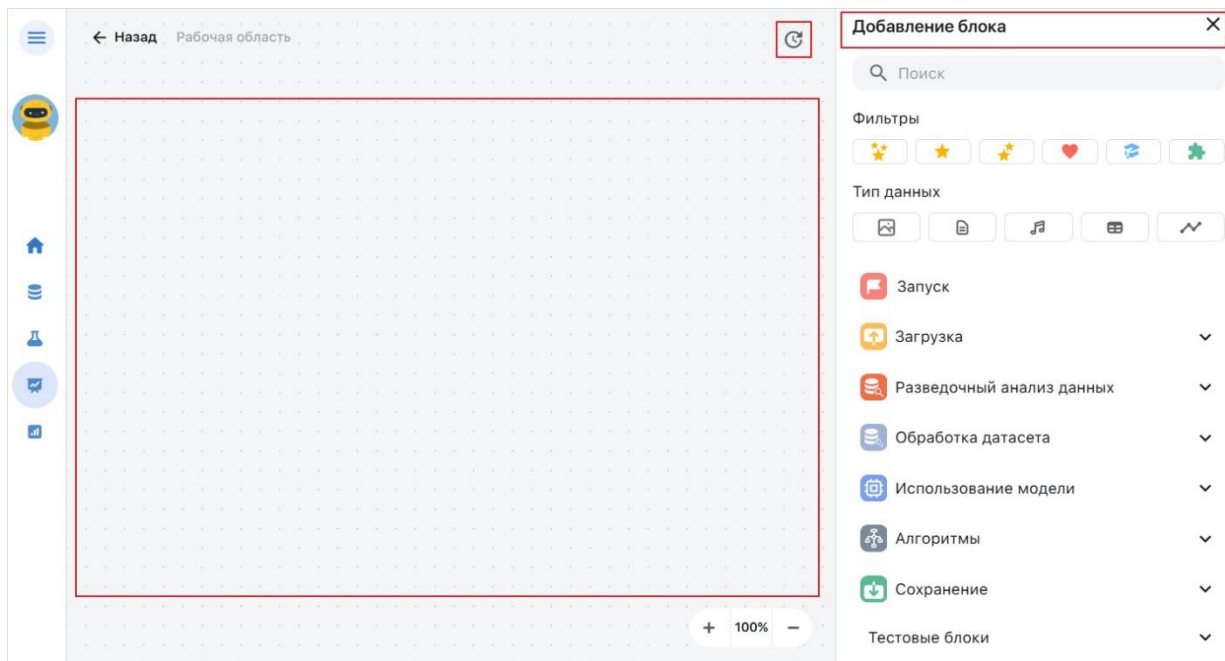
Процесс управления рабочими областями аналогичен тому, как в разделе «Обучение модели ИИ».



51.2 Обзор рабочей области

Интерфейс рабочей области для использования моделей аналогичен интерфейсу для обучения. Отличается набор доступных блоков.

- Основное поле, где происходит размещение и соединение блоков
- Панель добавления блоков
- История запусков (↺)



51.2.1. Панель добавления блоков

Набор блоков в разделе «Использование модели ИИ» адаптирован специально для задач инференса. Здесь отсутствуют блоки для обучения и валидации. Добавлены блоки для применения моделей.

Основные группы блоков:

- Загрузка данных и моделей
- Разведочный анализ данных (EDA)
- Обработка датасета
- Использование модели (ключевая группа с блоками применения)
- Алгоритмы
- Сохранение результатов прогнозирования

51.2.2. Процесс сборки пайплайна

Основные механики сборки пайплайна — добавление, перемещение, соединение и удаление блоков, а также работа с окнами визуализации — идентичны тем, что описаны в разделе «Обучение модели ИИ». В следующем разделе рассмотрены практические примеры построения таких пайплайнов.

52. Примеры использования моделей

В этом разделе рассмотрим, как использовать готовые модели для решения задач классификации, регрессии и кластеризации.

Типичный пайплайн использования модели состоит всего из трех логических шагов:

1. Загрузить новые данные.
2. Загрузить обученную модель.
3. Применить модель к данным и получить прогнозы.

52.1 Применение модели классификации

Рассмотрим, как использовать созданную модель для классификации вида животного на новых данных.

Задача

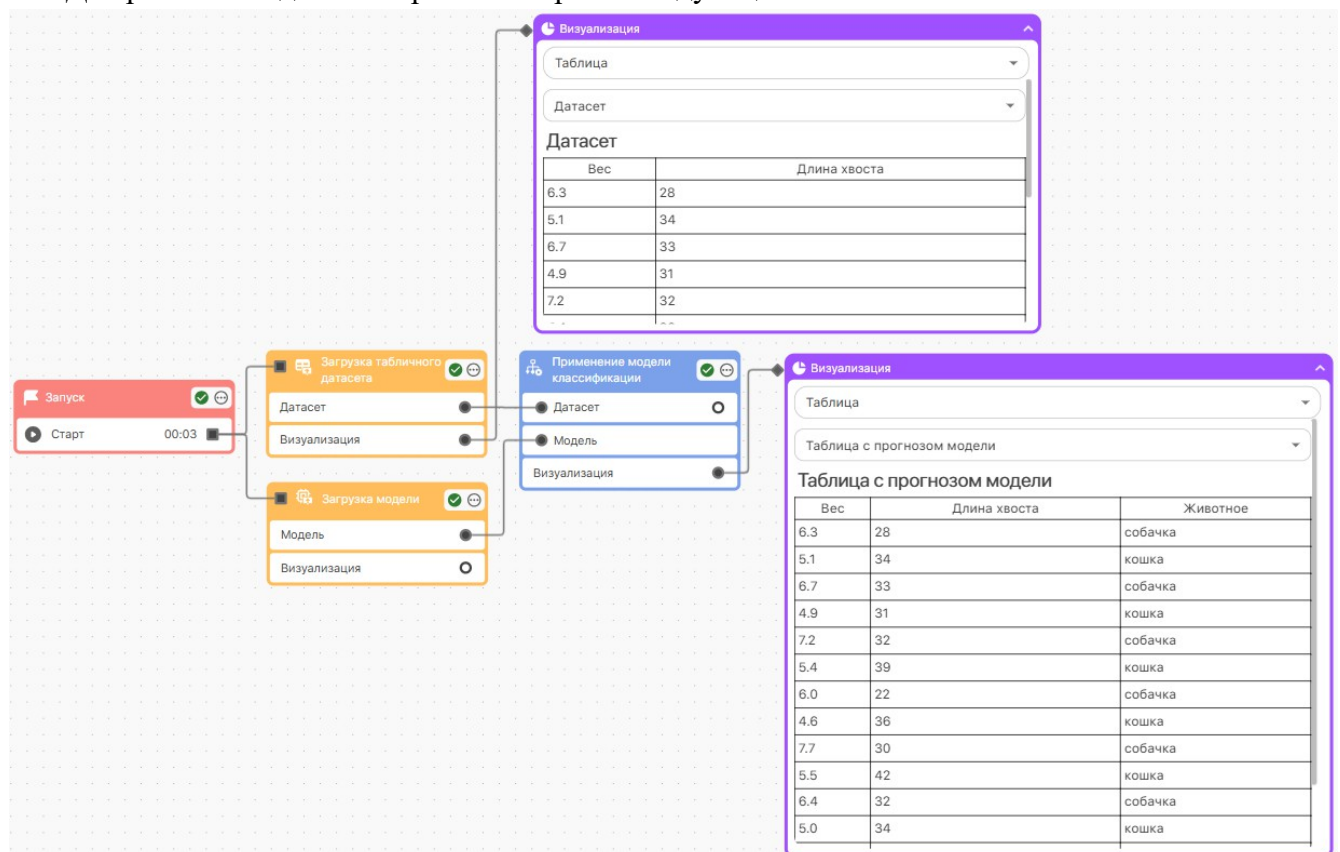
Определить вид животного (кошка или собака) на основе его веса и длины хвоста.

Входные данные

Для прогноза будем использовать тестовую выборку из предыдущего примера. В ней отсутствует целевой столбец Животное.

Построение пайплайна

Для решения задачи соберем и настроим следующий пайплайн:

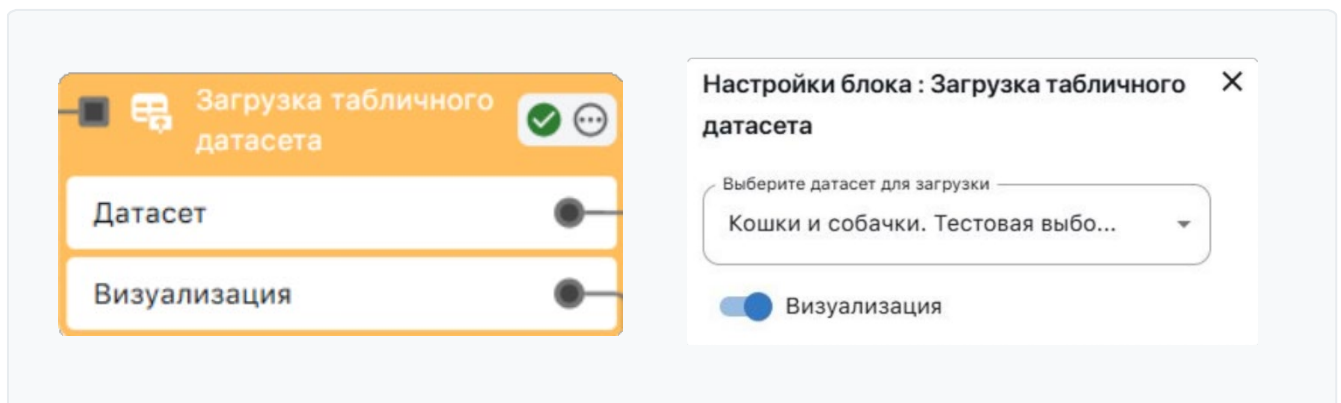


Шаг 1. Загрузка данных

Первый шаг — загрузить данные, для которых хотим сделать прогноз. Используем блок «Загрузка табличного датасета».

Настройка

В выпадающем списке «Выберите датасет для загрузки» указываем файл с новыми данными — «Кошки и собаки. Тестовая выборка».



Визуализация показывает загруженную таблицу с признаками — столбцами "Вес" и "Длина хвоста".

Визуализация

Таблица

Датасет

Датасет

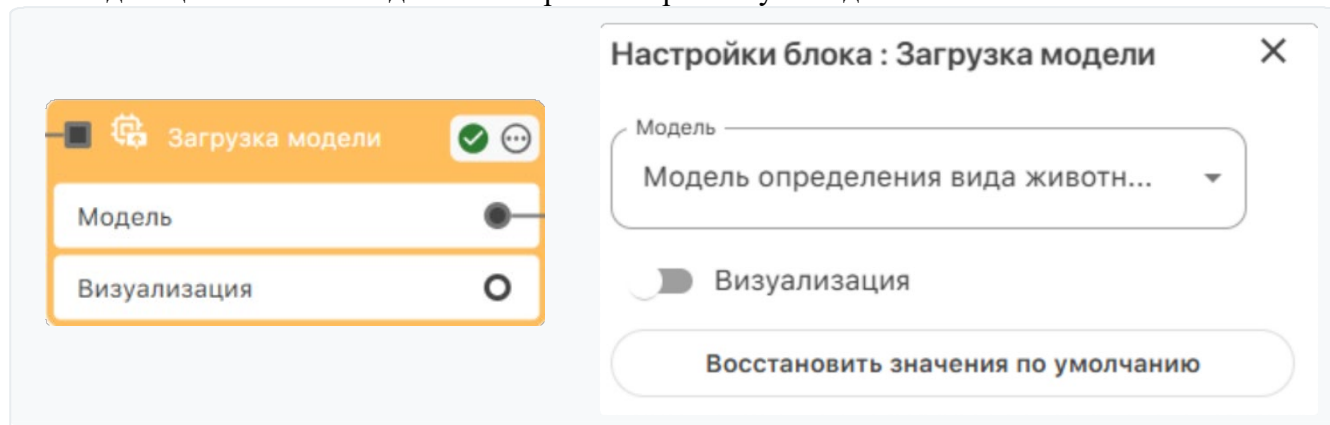
Вес	Длина хвоста
6.3	28
5.1	34
6.7	33
4.9	31
7.2	32
5.4	39
6.0	22
4.6	36
7.7	30
5.5	42
6.4	32
5.0	34
6.9	31
4.8	30
5.6	29

Шаг 2. Загрузка обученной модели

Теперь загрузим модель, которую обучили и сохранили на предыдущем этапе. Для этого используется блок «Загрузка модели».

Настройка

В выпадающем списке «Модель» выбираем сохраненную модель.

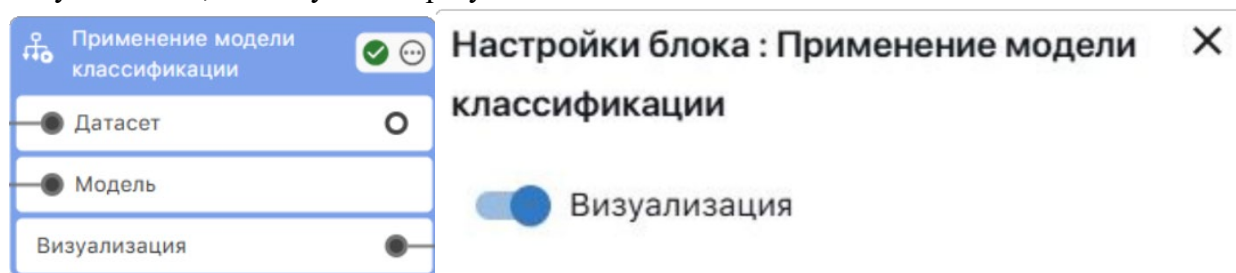


Шаг 3. Применение модели классификации и получение прогнозов

На этом шаге выполняется инференс — применение обученной модели к новым данным для получения прогнозов. Используем блок «Применение модели классификации».

Настройка

Блок не требует настроек. Достаточно просто подключить входы и активировать визуализацию, чтобы увидеть результат.



Анализ результата

На выходе получаем исходную таблицу с новым столбцом «Животное». Значения в этом столбце — прогнозы, сделанные моделью.

Визуализация

Таблица

Таблица с прогнозом модели

Таблица с прогнозом модели

Вес	Длина хвоста	Животное
6.3	28	собачка
5.1	34	кошка
6.7	33	собачка
4.9	31	кошка
7.2	32	собачка
5.4	39	кошка
6.0	22	собачка
4.6	36	кошка
7.7	30	собачка
5.5	42	кошка
6.4	32	собачка
5.0	34	кошка
6.9	31	собачка
4.8	30	кошка
5.6	29	собачка

Вывод: таким образом, была успешно применена ранее обученная модель для решения практической задачи на новых данных.

52.2 Применение модели регрессии

Рассмотрим, как использовать созданную модель регрессии для предсказания возраста новых животных.

Задача

Предсказать возраст на основе физических и поведенческих характеристик животного.

Входные данные

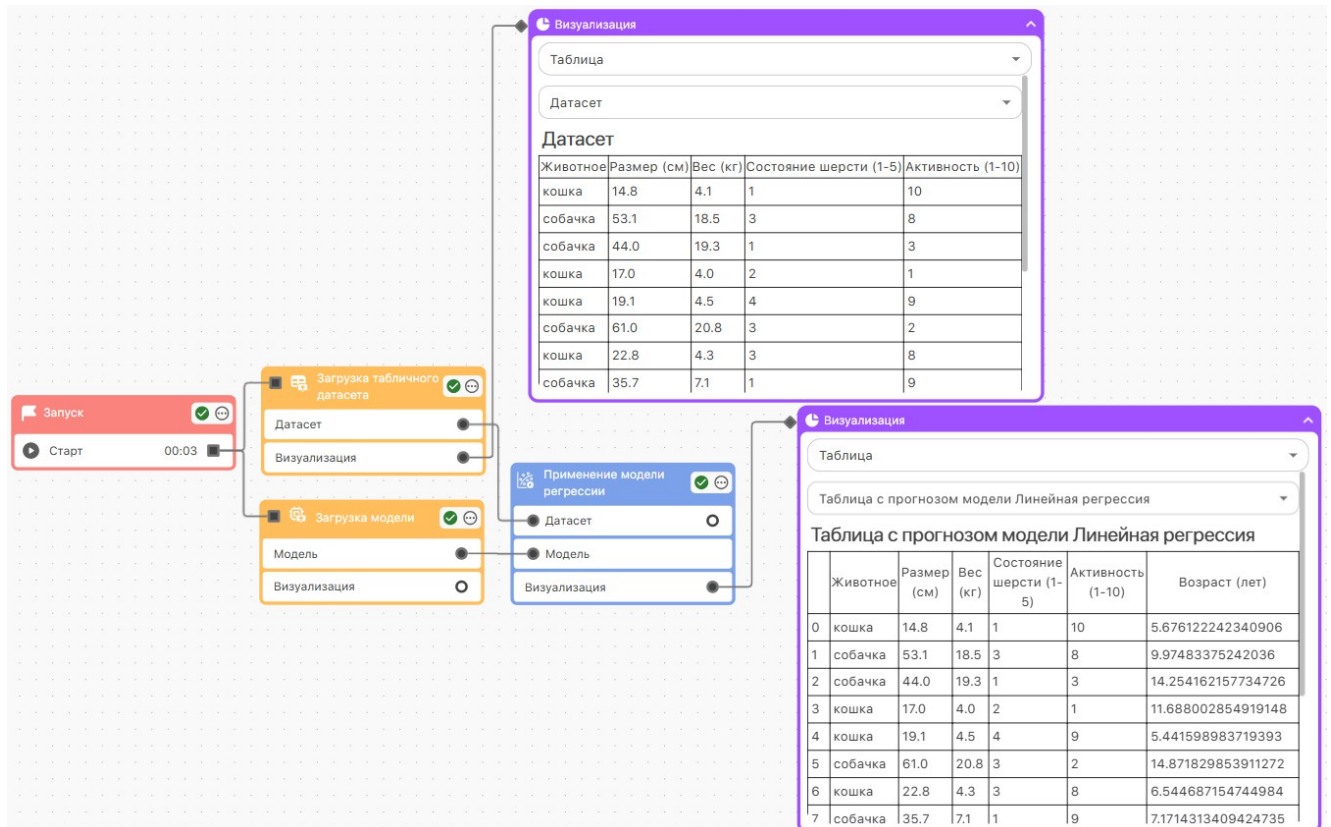
Для прогноза используем таблицу, в которой отсутствует целевой столбец «Возраст (лет)», но присутствуют все признаки, на которых обучалась модель:

- Животное
- Размер (см)
- Вес (кг)
- Состояние шерсти (1-5)

— Активность (1-10)

Построение пайплайна

Для решения задачи соберем пайплайн:

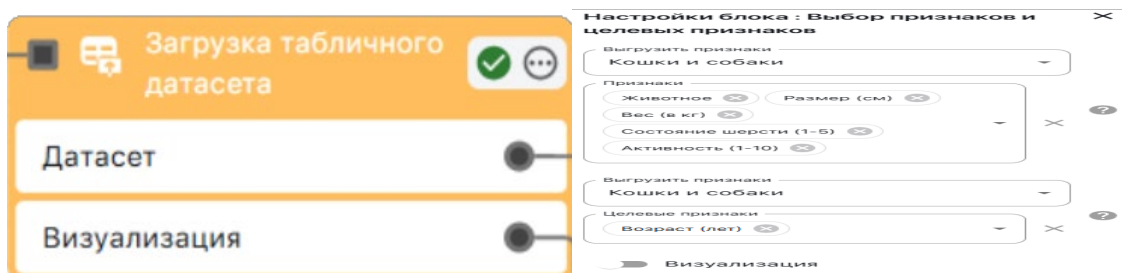


Шаг 1. Загрузка данных

Загрузим данные с помощью блока «Загрузка табличного датасета».

Настройка

В выпадающем списке «Выберите датасет для загрузки» указываем файл с новыми данными.



Визуализация ниже отображает загруженную таблицу со всеми необходимыми признаками.

Визуализация

Таблица

Датасет

Датасет

Животное	Размер (см)	Вес (кг)	Состояние шерсти (1-5)	Активность (1-10)
кошка	14.8	4.1	1	10
собачка	53.1	18.5	3	8
собачка	44.0	19.3	1	3
кошка	17.0	4.0	2	1
кошка	19.1	4.5	4	9
собачка	61.0	20.8	3	2
кошка	22.8	4.3	3	8
собачка	35.7	7.1	1	9
кошка	23.0	3.5	2	10
собачка	70.9	14.3	2	4
кошка	29.7	5.5	5	2
собачка	69.2	13.7	3	6
собачка	40.5	6.4	1	3
кошка	21.7	4.7	1	10
кошка	28.2	4.5	3	3

Шаг 2. Загрузка модели

Теперь загрузим модель, которую обучили и сохранили на предыдущем этапе. Для этого используется блок «Загрузка модели».

Настройка

В выпадающем списке «Модель» выбираем сохраненную модель регрессии.

Загрузка модели

Модель

Визуализация

Настройки блока : Загрузка модели

Модель

Модель регрессии

Визуализация

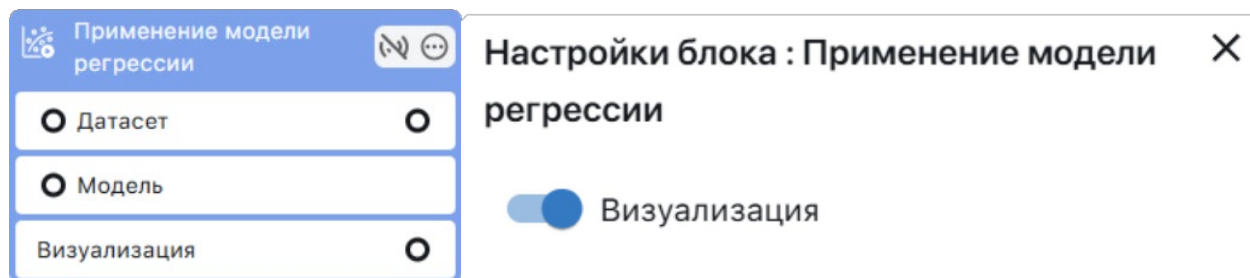
Восстановить значения по умолчанию

Шаг 3. Применение модели регрессии и получение прогнозов

На этом ключевом шаге выполняется инференс. Блок «Применение модели регрессии» берет данные с признаками и обученную модель, после чего для каждой строки вычисляет прогнозное значение возраста.

Настройка

Блок не требует настроек. Достаточно активировать визуализацию, чтобы увидеть итоговую таблицу с прогнозами.



Анализ результата

На выходе получаем исходную таблицу с добавленным столбцом «Возраст (лет)» (прогнозы).

Визуализация						
Таблица						
Таблица с прогнозом модели Линейная регрессия						
Таблица с прогнозом модели Линейная регрессия						
	Животное	Размер (см)	Вес (кг)	Состояние шерсти (1-5)	Активность (1-10)	Возраст (лет)
0	кошка	14.8	4.1	1	10	5.676122242340906
1	собачка	53.1	18.5	3	8	9.97483375242036
2	собачка	44.0	19.3	1	3	14.254162157734726
3	кошка	17.0	4.0	2	1	11.688002854919148
4	кошка	19.1	4.5	4	9	5.441598983719393
5	собачка	61.0	20.8	3	2	14.871829853911272
6	кошка	22.8	4.3	3	8	6.544687154744984
7	собачка	35.7	7.1	1	9	7.1714313409424735
8	кошка	23.0	3.5	2	10	5.328788673955521
9	собачка	70.9	14.3	2	4	12.585084000741032
10	кошка	29.7	5.5	5	2	10.440292448134267
11	собачка	69.2	13.7	3	6	10.636815018960153
12	собачка	40.5	6.4	1	3	11.349807529676449
13	кошка	21.7	4.7	1	10	5.9384896694378035
14	кошка	28.2	4.5	3	3	10.225463889875847

Вывод: успешно применена обученная модель регрессии для прогнозирования числового значения на новых данных. Результаты выглядят правдоподобно, с учетом дробных значений, характерных для задач регрессии.

52.3 Применение модели кластеризации

Рассмотрим, как использовать созданную модель кластеризации для разделения новых данных на группы.

Задача

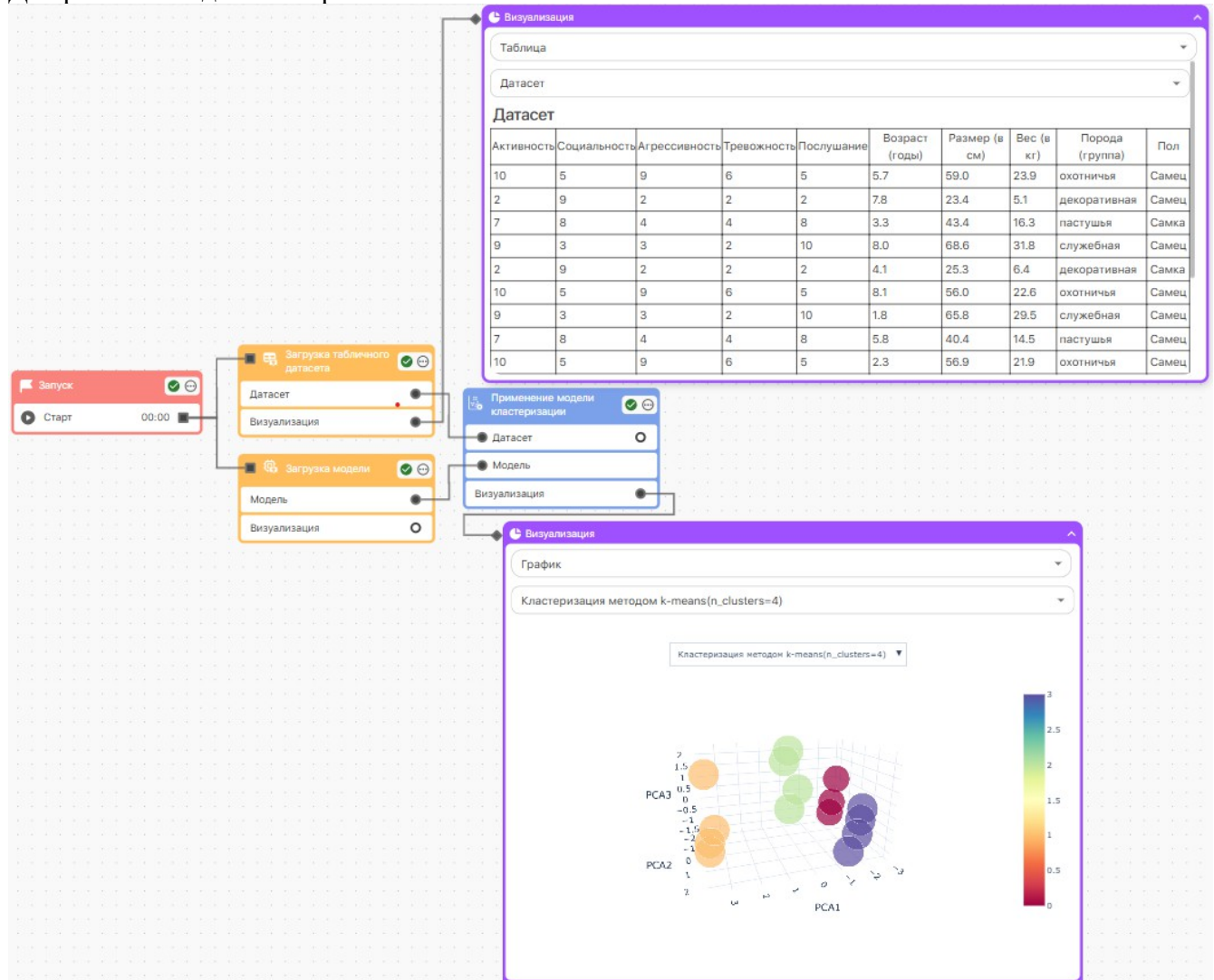
Присвоить каждой новой собаке метку кластера (от 0 до 3) на основе ее характеристик, используя ранее обученную модель.

Входные данные

Для прогноза используется тестовая таблица, содержащая все те же признаки, что и при обучении. Модель будет использовать все 10 признаков для определения наиболее подходящего кластера.

Построение пайплайна

Для решения задачи соберем пайплайн:

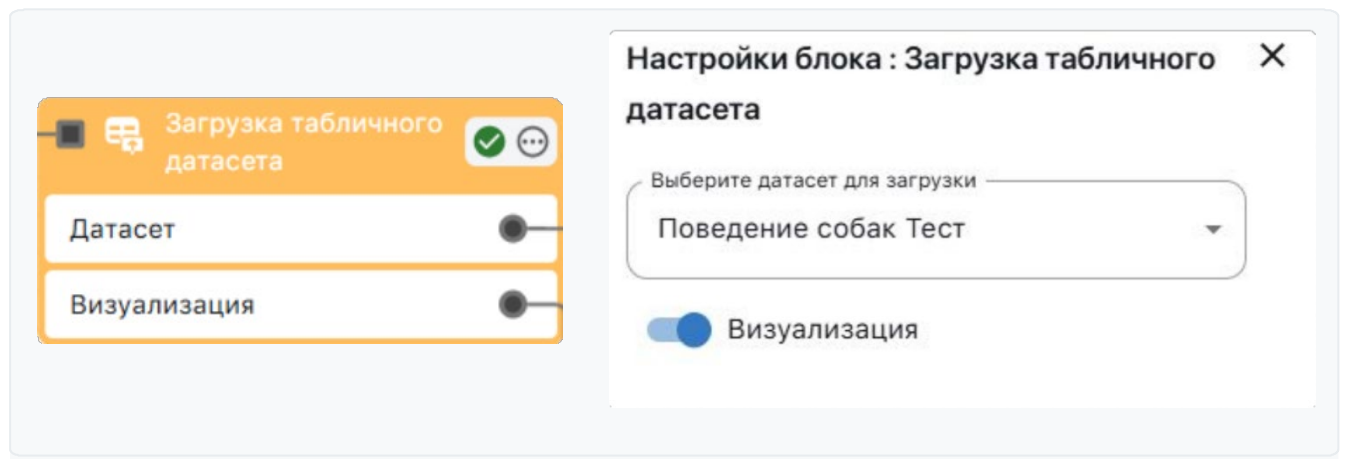


Шаг 1. Загрузка новых данных

Загрузим данные с помощью блока «Загрузка табличного датасета».

Настройка

В выпадающем списке «Выберите датасет для загрузки» указываем файл с новыми данными.



Визуализация показывает загруженную тестовую таблицу со всеми 10 признаками, необходимыми для работы модели.

Визуализация

Таблица

Датасет

Датасет

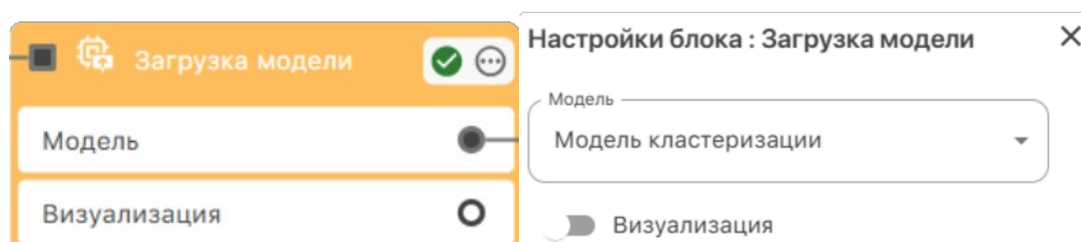
Активность	Социальность	Агрессивность	Тревожность	Послушание	Возраст (годы)	Размер (в см)	Вес (в кг)	Порода (группа)	Пол
10	5	9	6	5	5.7	59.0	23.9	охотничья	Самец
2	9	2	2	2	7.8	23.4	5.1	декоративная	Самец
7	8	4	4	8	3.3	43.4	16.3	пастушья	Самка
9	3	3	2	10	8.0	68.6	31.8	служебная	Самец
2	9	2	2	2	4.1	25.3	6.4	декоративная	Самка
10	5	9	6	5	8.1	56.0	22.6	охотничья	Самец
9	3	3	2	10	1.8	65.8	29.5	служебная	Самец
7	8	4	4	8	5.8	40.4	14.5	пастушья	Самец
10	5	9	6	5	2.3	56.9	21.9	охотничья	Самец
9	3	3	2	10	6.0	67.4	31.2	служебная	Самец
7	8	4	4	8	2.9	44.3	16.0	пастушья	Самец
2	9	2	2	2	5.9	24.7	6.6	декоративная	Самец
7	8	4	4	8	4.7	42.2	15.5	пастушья	Самка
10	5	9	6	5	3.8	58.0	23.0	охотничья	Самец
2	9	2	2	2	9.2	24.5	5.6	декоративная	Самец

Шаг 2. Загрузка обученной модели

Теперь загрузим модель, которую обучили и сохранили на предыдущем этапе с помощью блока «Загрузка модели».

Настройка

В выпадающем списке «Модель» выбираем сохраненную модель кластеризации.

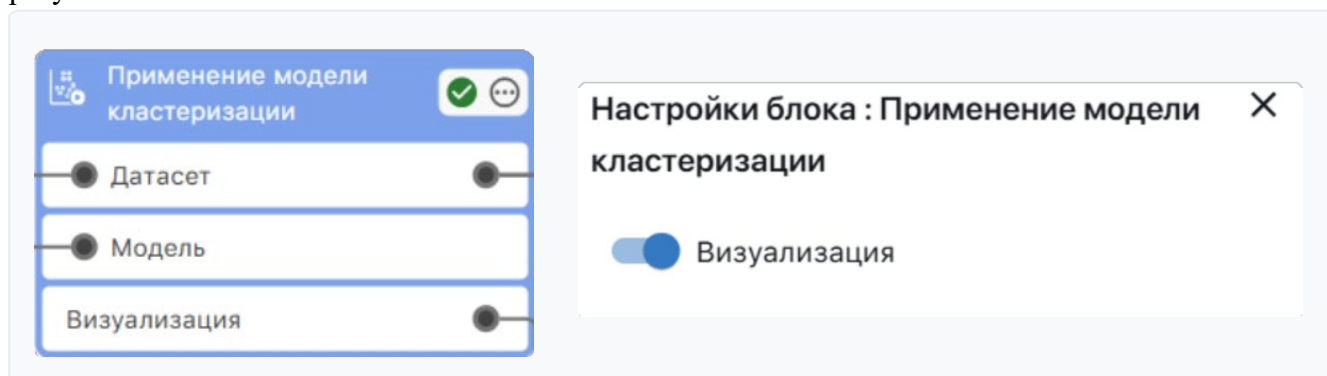


Шаг 3. Применение модели и анализ результатов

Выполним инференс с помощью блока «Применение модели кластеризации», который берет тестовые данные и обученную модель и для каждой строки определяет наиболее подходящий кластер.

Настройка

Блок не требует настроек. Достаточно активировать визуализацию, чтобы увидеть результат.



Анализ результата

В результате применения модели к тестовым данным формируется несколько визуализаций, позволяющих оценить качество её работы.

1. 3D и 2D визуализации кластеров

Графики показывают, что модель уверенно разделила все тестовые собаки на 4 четких и хорошо обособленных кластера. Каждая группа точек (кластер) плотно сгруппирована и находится на значительном расстоянии от других. Это говорит о высокой "уверенности" модели в своем решении и подтверждает, что она успешно распознала скрытые группы в новых данных.

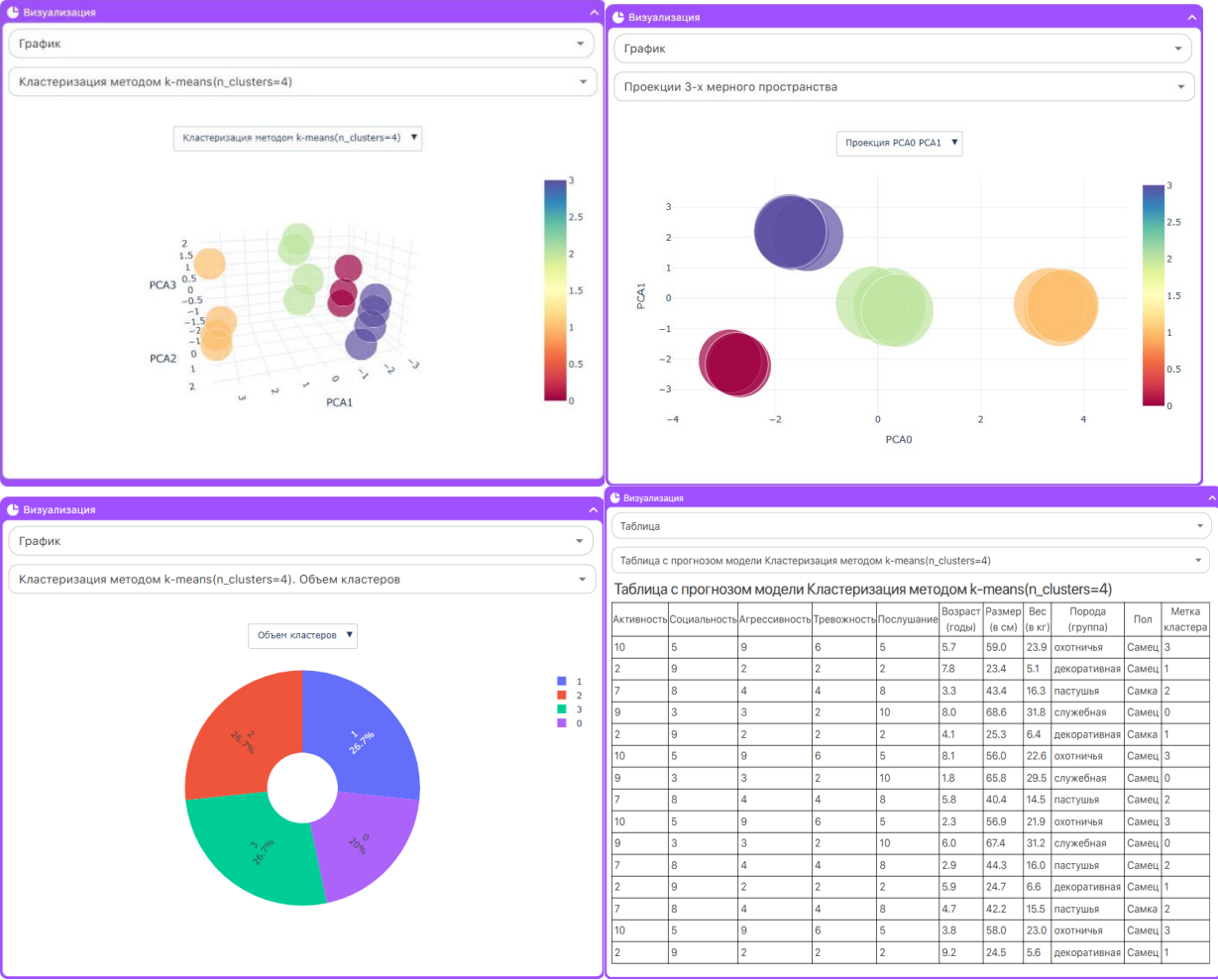
2. Объем кластеров (круговая диаграмма)

Диаграмма показывает, как тестовые собаки распределились по найденным группам. Размеры кластеров получились сбалансированными (от 20% до 26.7%), что соответствует составу тестового набора данных и подтверждает, что модель не создала один доминирующий кластер, а равномерно распределила объекты.

3. Таблица с прогнозом (с метками кластеров)

Это итоговый результат. В таблице к исходным данным добавлен новый столбец «Метка кластера». Каждое число в этом столбце (0, 1, 2 или 3) — номер группы, к которой модель отнесла данную собаку. Сравнивая метки с исходным столбцом «Порода (группа)», можно убедиться, что модель сработала отлично:

- Все "служебные" собаки получили метку 0.
- Все "декоративные" — метку 1.
- Все "пастушьи" — 2.
- Все "охотничьи" — 3.



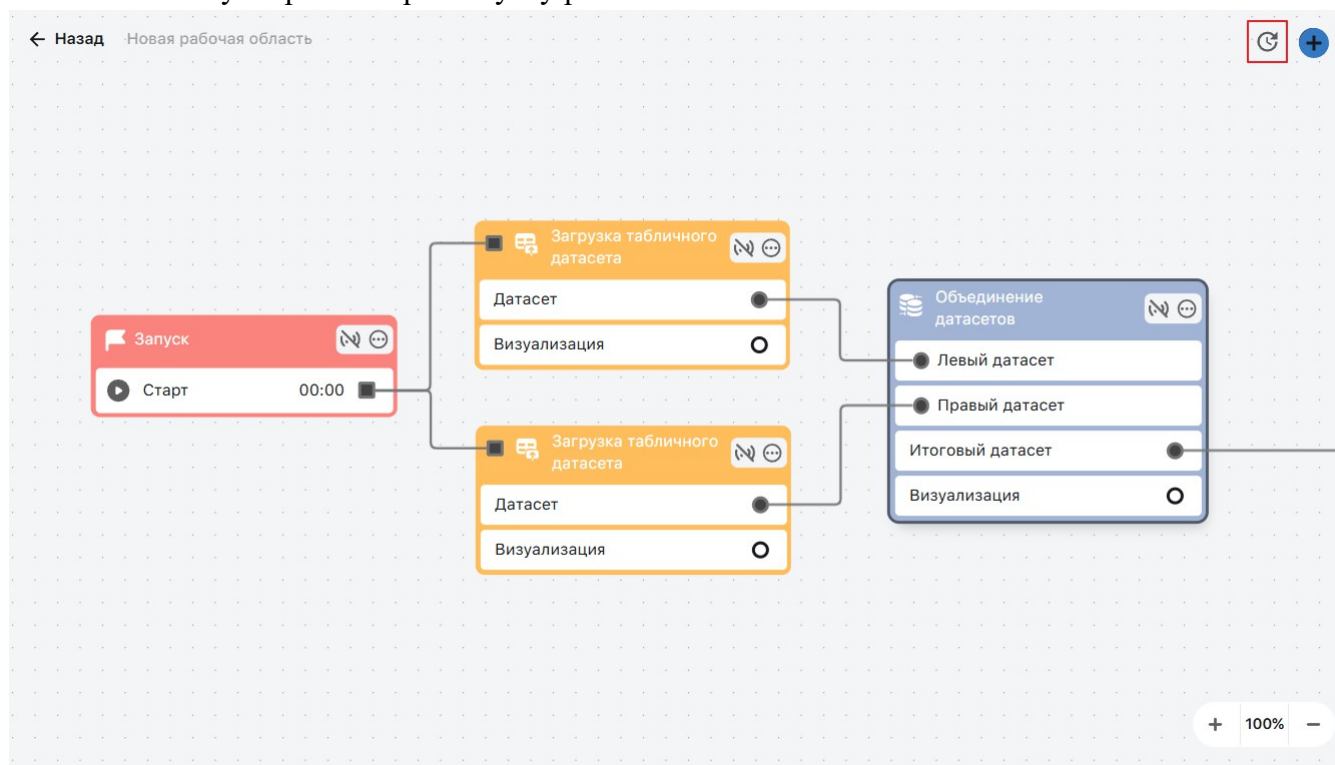
Вывод: модель кластеризации отлично справилась с задачей. Она присвоила каждой новой собаке метку кластера с высокой точностью, что подтверждается как четким визуальным разделением на графиках, так и полным соответствием меток реальным группам пород в итоговой таблице. Это демонстрирует, что хорошо обученная модель может эффективно и осмысленно сегментировать новые, ранее не виденные данные.

53. Порядок работы с коннекторами

Коннекторы — это именованные точки доступа к результатам (артефактам), сгенерированным в ходе выполнения пайплайна. Они служат мостом между результатами работы блоков и дашбордом.

Коннекторы позволяют визуализировать данные на дашборде. Каждый коннектор соответствует определенному типу данных, например, таблице, графику, тексту или изображению. Число коннекторов определяется блоком исходя из логики решения задачи.

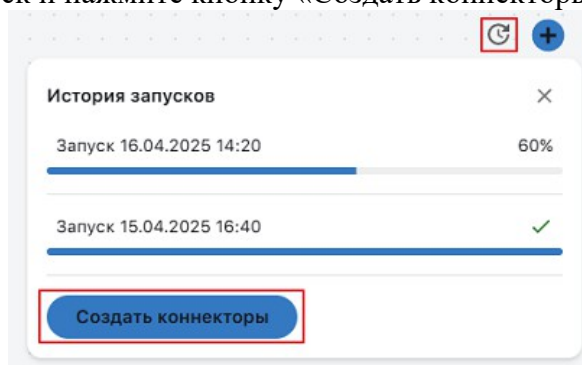
После того как ваш пайплайн успешно завершил выполнение, откройте историю запусков, нажав на кнопку в правом верхнем углу рабочей области.



В окне «История запусков» найдите нужный запуск. Статус выполнения отображается следующим образом:

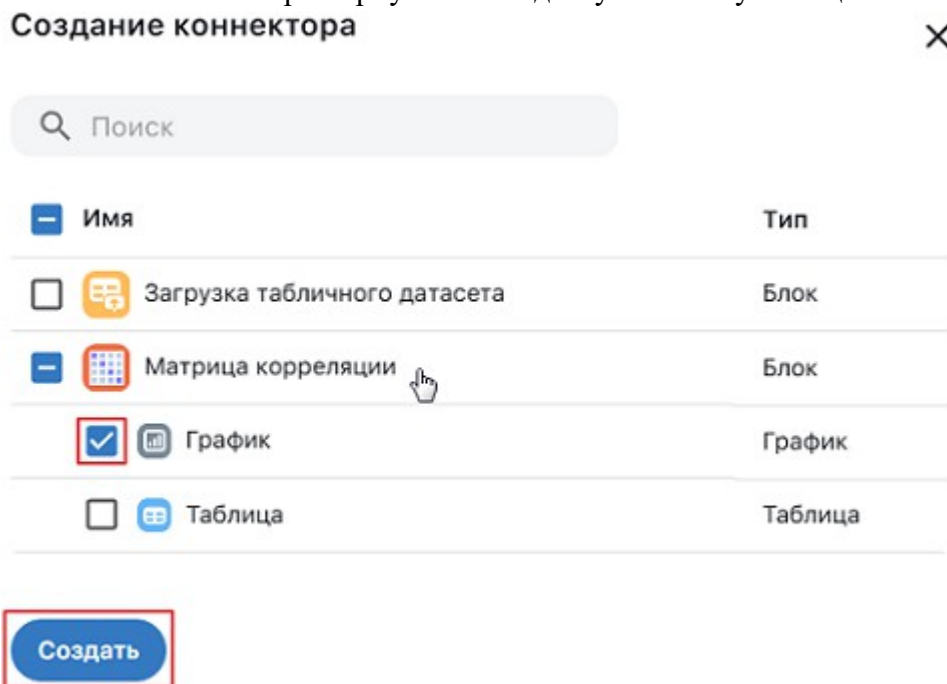
- Зеленая галочка — пайплайн завершен успешно.
- Индикатор с процентами — отображает процент выполнения для незавершенных или ошибочных запусков.

Выберите успешный запуск и нажмите кнопку «Создать коннекторы».



Важно: Создать коннекторы возможно только для полностью и успешно завершенных запусков.

В модальном окне «Создание коннектора» отобразится список всех блоков пайплайна. При нажатии на названия блоков развернутся типы доступных визуализаций:



Установите флажки напротив тех блоков, результаты которых хотите визуализировать на дашборде. Можно выбрать все или только некоторые. Нажмите на кнопку «Создать».

Если все успешно, вы увидите уведомление «Коннекторы созданы».

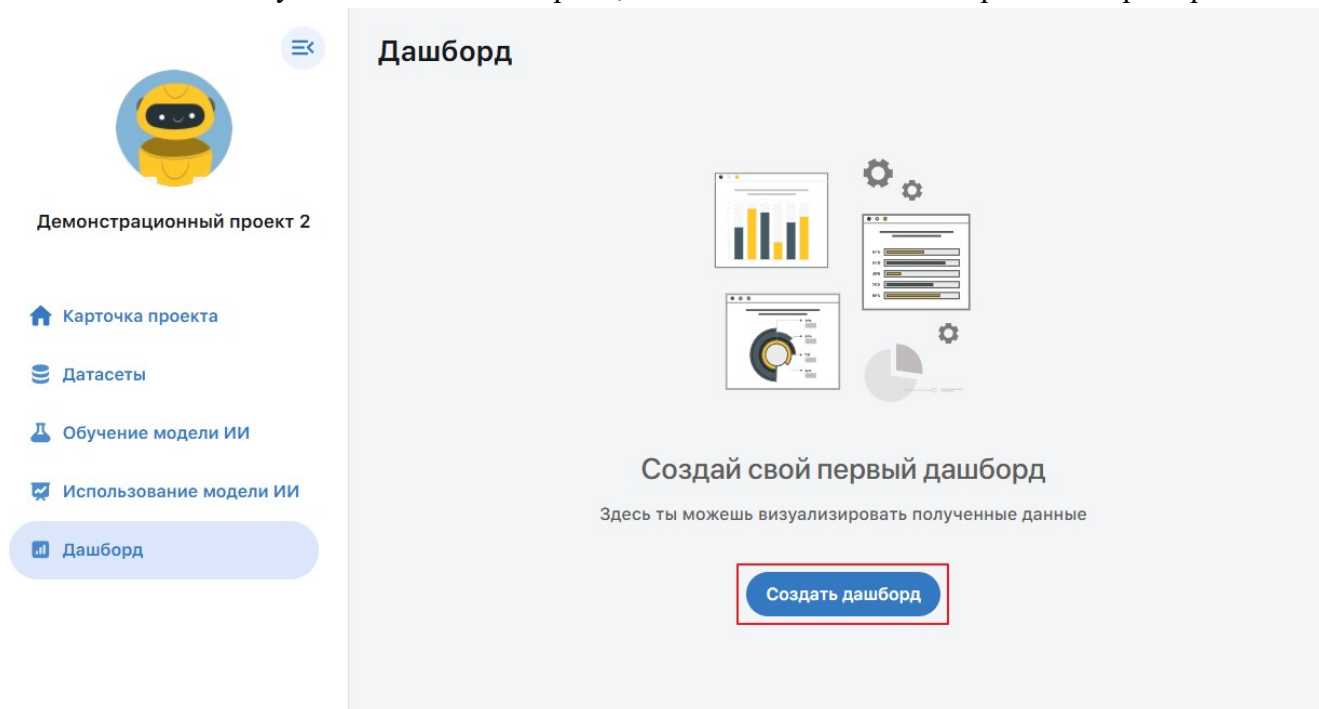
Теперь доступны в качестве источников данных для добавления на дашборд. Вы можете переходить в раздел «Дашборд», чтобы использовать их для построения виджетов и визуализации полученных результатов.

54. Порядок работы с дашбордами

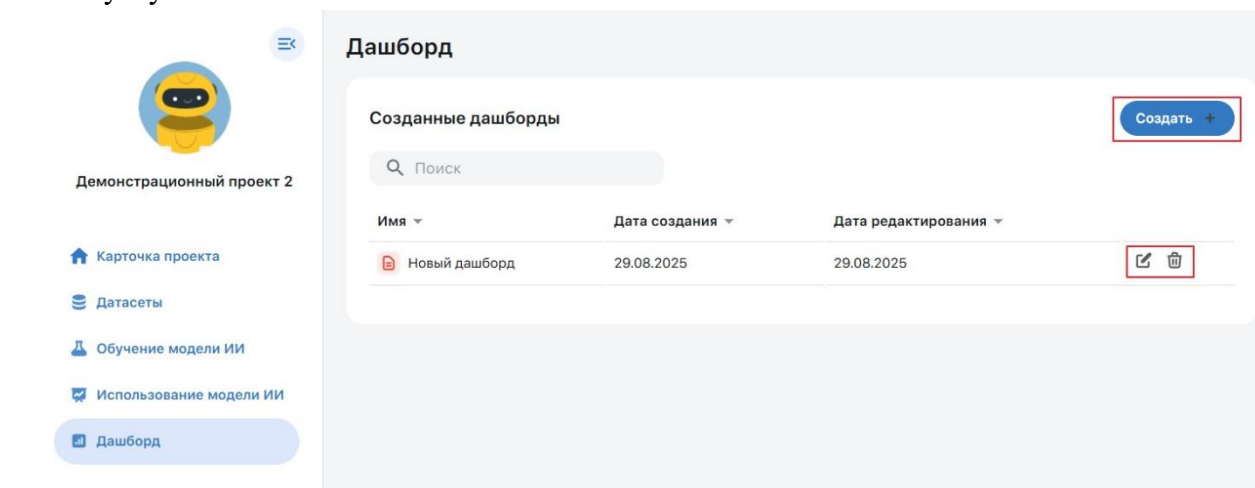
Дашборд — это интерактивная рабочая область для наглядного представления и визуализации данных. На дашборд можно выводить результаты обучения или использования моделей ИИ в виде графиков, таблиц и других блоков для комплексного анализа проекта.

Перейдите в раздел «Дашборд» в левом меню проекта.

- Если у вас еще нет дашбордов, нажмите «Создать дашборд» в центре экрана.



- Если дашборды уже существуют, используйте «Создать +» в правом верхнем углу.



Для наполнения дашборда необходимо предварительно создать коннекторы из успешных запусков пайплайнов. Подробнее о работе с коннекторами вы можете посмотреть в разделе Порядок работы с коннекторами.

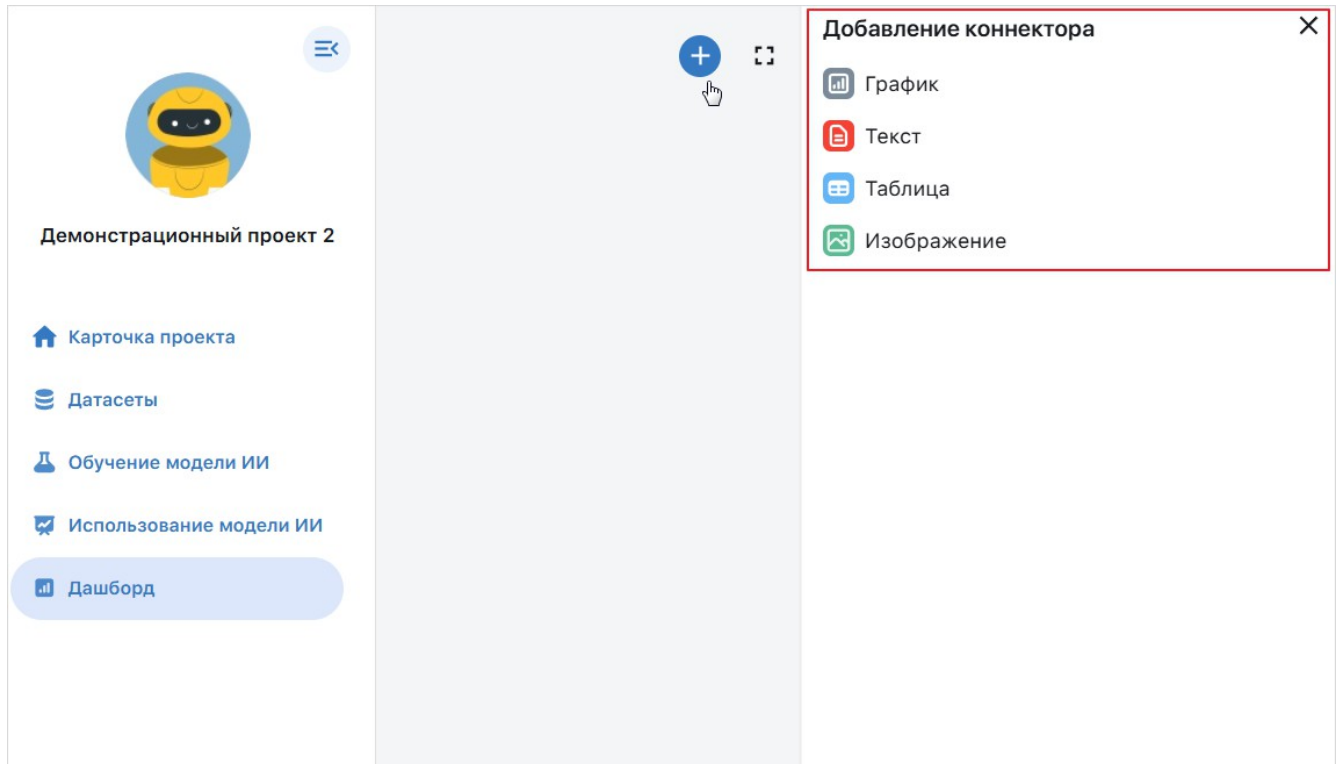
54.1 Управление виджетами на рабочей области

После создания дашборда вы попадете на пустую рабочую область.

Чтобы добавить виджет:

Нажмите на синюю кнопку «+» в правом верхнем углу, чтобы открыть меню «Добавление коннектора».

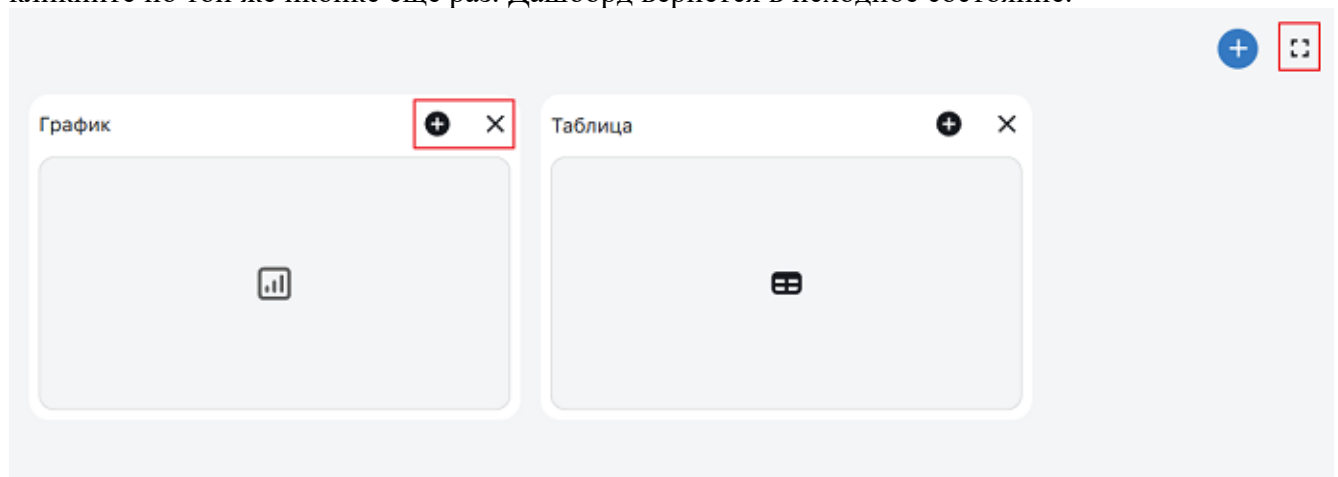
Выберите нужный тип блока (График, Текст, Таблица или Изображение) и кликните по нему.



На рабочей области появится пустой виджет выбранного типа.

Чтобы удалить виджет, нажмите на крестик в правом верхнем углу блока.

Чтобы перейти в полноэкранный режим, нажмите на соответствующую иконку в правом верхнем углу рабочей области (панели управления). Для выхода нажмите клавишу Esc или кликните по той же иконке еще раз. Дашборд вернется в исходное состояние.



54.2 Выбор коннектора для виджета

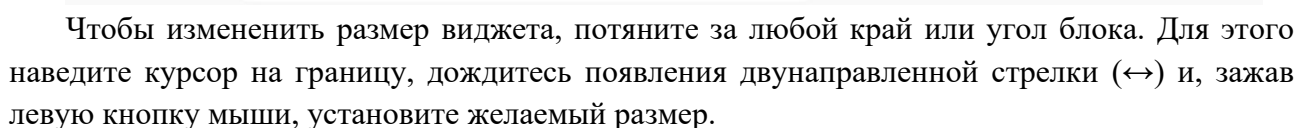
Теперь нужно «оживить» виджет, подключив к нему источник данных — коннектор. В правом верхнем углу добавленного блока нажмите на «+» и выберите необходимый

Установите флажок напротив нужного коннектора и нажмите кнопку «Сохранить».

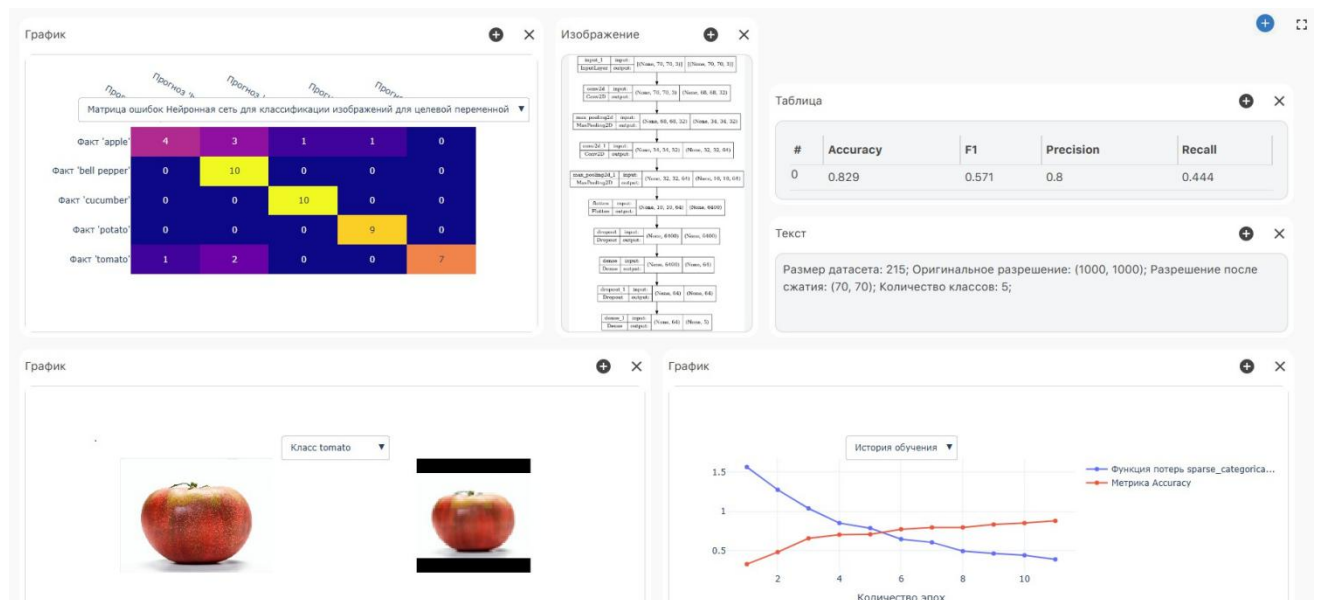
Виджет отобразит данные из выбранного коннектора. Повторите эти шаги для всех виджетов, которые вы хотите видеть на дашборде.

Виджеты на дашборде можно свободно перемещать и изменять их размер, чтобы создать удобное для вас представление.

В процессе вы увидите полупрозрачный фиолетовый плейшолдер, который показывает, где будет размещен виджет. Отпустите кнопку мыши, чтобы зафиксировать его на новом месте.



576



График

Класс: tomato

График

История обучения

Функция потерь sparse_categorical_crossentropy

Метрика Accuracy

1.5

1

0.5

2

4

6

8

10

Количество эпох