ARENADATA
Корпоративная платформа хранения
и обработки больших данных

Общество с ограниченной ответственностью
«Аренадата Софтвер»

Описание технической архитектуры программного обеспечения для электронно-вычислительных машин Arenadata Advanced RAG (ARAG)

Москва
2025

1 Оглавление

Термины и определения

Термин	Значение
ANSI SQL	Стандартный язык запросов к базам данных, разработанный Американским национальным стандартом (ANSI)
LLM-модели	Языковые модели, обученные на больших объемах текстовых данных

Сокращения и обозначения

Сокращение	Наименование
API	(англ. – Application Programming Interface) – набор классов, процедур, функций, структур или констант, которыми одна компьютерная программа может взаимодействовать с другой программой
CPU	(англ. – Central Processing Unit) – центральный процессор
IP	(англ. – Internet Protocol) – маршрутизируемый протокол сетевого уровня стека TCP/IP
LLMs	Large language models
MSSQL	Microsoft SQL Server
SQL	(англ. – Structured Query Language) – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционных базах данных
БД	База данных
ИИ	Искусственный интеллект
ПО	Программное обеспечение
СУБД	Система управления базами данных

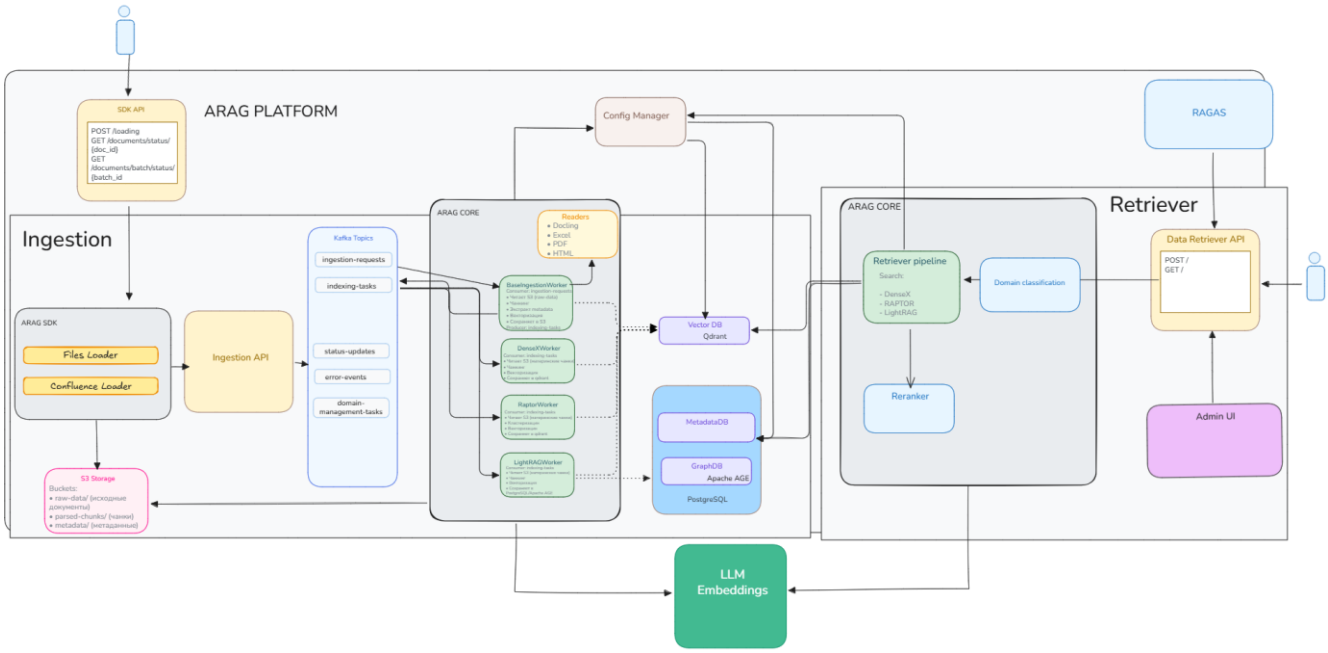
Arenadata Advanced RAG (ARAG) – это продвинутая платформа смыслового поиска (Retrieval Augmented Generation), которая помогает ИИ и людям находить точные ответы на специфичные для компании вопросы с учетом внутренних данных, регламентов и специфики отрасли.

ARAG сочетает различные стратегии векторного поиска, обеспечивая максимальную полноту и точность извлечения знаний. Модульная архитектура позволяет выборочно применять наиболее подходящие инструменты обработки информации для групп документов, отдельных источников данных или целых доменов знаний, а также выбирать наиболее подходящие для задачи ИИ-модели.

Платформа спроектирована для интеграции в корпоративные среды и масштабируется в зависимости от бизнес-сценариев.

Основные функции ПО:

- Семантический поиск – находит текст, близкий по смыслу к запросу, даже если нет точного совпадения ключевых слов;
- Поиск релевантных документов – извлекает информацию из внешних источников;
- Фильтрация и ранжирование – отбирает наиболее подходящие фрагменты текста по релевантности;
- Поддержка разных форматов – работает с файлами разных форматов, имеет возможность интеграции с источниками данных по API;
- Контекстно-зависимые ответы – использует найденные документы для генерации точных и развернутых ответов;
- Обобщение информации – может суммировать длинные документы или несколько источников;
- Ссылки на источники – сохраняет мета-данные исходного документа;
- Мультиязычность – поддерживает поиск и генерацию на разных языках.



Верхнеуровневая архитектурная схема

2 Ключевые архитектурные принципы

Представленная архитектура относится к распределенной микросервисной системе, построенной на принципах событийно-ориентированного подхода (Event-Driven Architecture, EDA).

Этот подход позволяет создать гибкую и масштабируемую платформу для построения продвинутых RAG-решений.

1. Событийно-ориентированная архитектура загрузки данных (Event-Driven Architecture)

Асинхронное взаимодействие: Компоненты системы общаются не напрямую, а через асинхронные события.

Слабая связанность (Loose Coupling): Сервисы (воркеры) не зависят друг от друга. Это позволяет добавлять новые механизмы обработки данных без изменения архитектуры системы.

Отказоустойчивость: Временный отказ одного из воркеров не останавливает всю систему. Задачи для него накапливаются в очереди Kafka и обработываются после восстановления.

2. Разделение ответственности (Separation of Concerns)

Специализация: Каждый компонент (API-шлюз, воркер) выполняет одну четко определенную задачу (например, прием запросов, базовый чанкинг, иерархическая индексация).

Независимое масштабирование: при увеличении нагрузки можно увеличить количество соответствующих воркеров.

3. Централизованное хранение загруженных (raw) данных

Объектное хранилище (S3/MinIO): Позволяет делать перепроцессинг без повторной загрузки.

3 Структура пакетов (модулей)

Архитектура кода разделена на логические уровни, что обеспечивает переиспользование и строгие границы ответственности.

1. `arag-platform` (Фундаментальный слой)

Назначение: Низкоуровневая инфраструктурная библиотека.

Состав: Готовые асинхронные клиенты для взаимодействия с PostgreSQL, Kafka и S3; система конфигурации на базе `pydantic-settings`; общие схемы данных (Pydantic-модели) и кастомные исключения.

Зависимости: Не имеет зависимостей от других пакетов проекта `arag`.

2. `arag-core` (Ядро RAG-логики)

Назначение: Здесь инкапсулирована вся предметно-ориентированная логика, необходимая для RAG-пайплайнов.

Состав: Модули для чанкинга (`TableAwareSentenceSplitter`), управления эмбеддингами (`EmbeddingManager`), взаимодействия с LLM (`LLMBase`), чтения файлов (`ReaderFactory`) и работы с векторными базами данных (`VectorStoreManager`).

Зависимости: Зависит только от `arag-platform`.

3. `arag-sdk` (Клиентский SDK)

Назначение: интерфейс для загрузки документов и отслеживания их статуса.

Состав: `ARAGClient` — основной фасад, который инкапсулирует логику загрузки файлов в S3 и отправки HTTP-запросов к `ingestion-api`.

Зависимости: Зависит только от `arag-platform`.

4. `ingestion-api` (API-шлюз)

Роль: API-шлюз, служащий единой точкой входа в систему.

Задачи:

- Принимать HTTP-запросы от `arag-sdk` на создание, обновление или удаление документов;
- Валидировать входящие запросы с помощью схем из `arag-platform`;
- Создавать первичную запись о документе в `MetadataDB` (PostgreSQL) с начальным статусом `uploaded`;
- Публиковать событие `IngestionEvent` в топик Kafka `ingestion-requests`.

5. `workers` (Воркеры)

Воркеры — это независимые микросервисы, которые выполняют основную вычислительную работу.

6. RAGAS

RAGAS (RAG Assessment) — это открытый Python-фреймворк, предназначенный для автоматической оценки качества RAG-систем. Он помогает измерить эффективность поиска на основе эталонного набора вопросов / ответов (Golden Questions) к загруженному дата-сету. Набор вопросов / ответов должен быть подготовлен заранее.

7. Admin UI

Техническая консоль позволяет выполнять как загрузку, так и поиск.

При загрузке новых документов, они загружаются в указанный домен. При поиске – поиск будет осуществлен в рамках указанного домена. Если домен не

указан, поиск осуществляется по домену, найденному с помощью алгоритма на основе ключевых слов.

При поиске на панели Pipeline Stages отображается техническая информация о всех метаданных о чанках.

4 Общая архитектура уровня ARAG Platform

ARAG Platform — это фундаментальный инфраструктурный слой всей системы ПО ARAG. Этот пакет не содержит бизнес-логики, специфичной для RAG (например, чанкинга или создания эмбеддингов). Его единственная цель — предоставить компоненты для взаимодействия с внешней инфраструктурой. Он инкапсулирует сложность работы с БД, брокерами сообщений и объектными хранилищами, предоставляя остальным частям системы чистый и унифицированный API.

Ключевые принципы уровня ARAG Platform:

- Нулевая бизнес-логика: Пакет содержит только «технические» компоненты: клиенты, схемы данных, конфигурацию и исключения;
- Переиспользуемость: Любой сервис в экосистеме ARAG (`ingestion-api`, `workers`, `arag_sdk`) использует этот пакет для взаимодействия с инфраструктурой, что исключает дублирование кода;
- Централизованный «контракт данных»: Модуль `schemas` определяет Pydantic-модели для сущностей в системе (события Kafka, документы, запросы/ответы API);
- Конфигурируемость: Управление конфигурацией всей системы централизовано через `arag_platform.config`, что позволяет настраивать компоненты отдельно;

Описание модулей ARAG Platform:

- Модуль `arag_platform.config` отвечает за централизованное управление конфигурацией всей системы;
- Модуль `arag_platform.database` предоставляет асинхронный клиент для взаимодействия с базой метаданных PostgreSQL;

- Модуль `arag_platform.messaging` содержит клиенты для работы с брокером сообщений Apache Kafka;
- Модуль `arag_platform.storage` отвечает за взаимодействие с S3-совместимым объектным хранилищем;
- Модуль `arag_platform.schemas` определяет все «контракты данных» в системе с помощью Pydantic. Назначение: Этот модуль — один из самых важных в `arag-platform`. Он обеспечивает консистентность данных между всеми сервисами;
- Модуль `arag_platform.exceptions` определяет иерархию кастомных исключений для стандартизированной обработки ошибок.

5 Уровень `arag_core` (Ядро RAG-логики)

Этот пакет содержит всю основную бизнес-логику, необходимую для преобразования сырых документов в индексируемые и готовые к поиску данные. В отличие от `arag-platform`, который занимается инфраструктурой, и `arag-sdk`, который является клиентским интерфейсом, `arag-core` фокусируется на RAG-пайплайне.

Этот уровень спроектирован как библиотека, которую используют воркеры (`workers`). Он не запускается как самостоятельный сервис, а предоставляет классы и функции, реализующие конкретные этапы обработки документов, такие как чтение, разделение на части (чанкинг), извлечение метаданных, создание векторных представлений (эмбеддингов) и сохранение в векторную базу данных.

Ключевые архитектурные принципы уровня `arag-core`:

- Инкапсуляция RAG-логики: Все алгоритмы и стратегии, связанные с обработкой текста и векторов, находятся здесь. Это позволяет обновлять и расширять RAG-возможности системы, не затрагивая инфраструктурные компоненты;

- Зависимость от `arag-platform`: `arag-core` использует `arag-platform` для получения конфигурации и взаимодействия с базовыми библиотеками, но не реализует клиенты для S3, Kafka или PostgreSQL самостоятельно;
- Реализован на `LlamaIndex`: Многие компоненты в `arag-core` являются обертками, расширениями или кастомизированными версиями классов из библиотеки `LlamaIndex`.
- Независимость от сервисов: `arag-core` – не содержит логики работы с Kafka или HTTP. Его компоненты — это строительные блоки, которые воркеры собирают в полноценный пайплайн обработки.

Описание модулей уровня `arag-core`:

- Модуль `arag_core.readers` отвечает за чтение и парсинг различных форматов файлов;
- Модуль `arag_core.chunking` содержит логику разделения текста на чанки;
- Модуль `arag_core.llm` предоставляет унифицированный интерфейс для работы с большими языковыми моделями (LLM);
- Модуль `arag_core.extractors` управляет извлечением дополнительной метаинформации из текстовых чанков;
- Модуль `arag_core.embeddings` отвечает за создание векторных представлений (эмбеддингов) для текста;
- Модуль `arag_core.vectorstore` содержит компоненты для взаимодействия с векторной базой данных Qdrant.

6 Уровень `arag_sdk` (Software Development Kit)

`arag_sdk` — это клиентская библиотека (SDK), предназначенная для разработчиков, которые хотят интегрировать свои приложения с ПО ARAG. Он предоставляет высокоуровневый, удобный и асинхронный интерфейс, скрывая всю сложность взаимодействия с внутренними компонентами системы.

Основная цель `arag_sdk` — обеспечить загрузку документов, управление ими и отслеживание их статуса в распределенной, событийно-ориентированной системе ARAG.

Ключевые архитектурные принципы уровня arag_sdk:

1. Модульность:

- Клиентский модуль: Главный фасад, с которым работает вызывающий сервис;
- Коммуникации: Клиент для взаимодействия с `ingestion-api` по HTTP;
- Хранение: Абстракция для работы с файловым хранилищем S3;
- Данные: Строгие модели данных, обеспечивающие консистентность.

2. Асинхронность: Все операции ввода-вывода (HTTP-запросы, работа с файлами в S3) реализованы асинхронно.

6.1 `ingestion-api` (API-шлюз)

Приложение FastAPI, которое предоставляет RESTful API для приема данных.

6.2 Воркеры

Набор потребителей Kafka, которые выполняют фактическую обработку данных.

7 Реализация Ragas

Модуль `ai_evaluation` предназначен для оценки производительности ПО ARAG с использованием фреймворка Ragas. Он автоматизирует процесс оценки, от загрузки данных до генерации отчетов.

Интеграция с ARAG

- Вызов через API: Основная связь осуществляется через `AdvancedRagConnector`;

- Оценка реальной производительности: модуль получает ответы и контексты, сгенерированные работающим ПО ARAG, и затем оценивает их качество в соответствии с рядом метрик;
- Общая конфигурация: Модуль использует тот же файл `config.yaml`, что и основное приложение. Это гарантирует, что оценка проводится с использованием тех же моделей (LLM, эмбеддинги) и настроек, что и в рабочем окружении, делая результаты оценки максимально релевантными.

Таким образом, модуль `ai_evaluation` функционирует как внешний инструмент для аудита и тестирования производительности всего ПО ARAG, имитируя обращения конечного пользователя к ее API.

8 Используемые сервисы

8.1 Топики Kafka

Для загрузки данных используется шина данных Kafka. `ingestion-api` публикует события в Kafka, а независимые воркеры их обрабатывают. Имена топиков настраиваются в файле `arag_sdk/core/config.py`. Схемы событий определены в `arag_sdk/schemas/kafka_events.py`.

8.2 Структура хранения в S3

Система использует структурированный подход к хранению загруженных (raw) данных в S3-совместимом хранилище. Имена префиксов (директорий) настраиваются в файле `arag_sdk/core/config.py` в классе `S3Settings`.

8.3 Векторная база данных Qdrant

В качестве базы для хранения векторов используется Qdrant.
Основные сущности Qdrant:

- Коллекция — это именованный набор точек (векторов с полезной нагрузкой), среди которых можно выполнять поиск. Векторы всех точек в пределах одной коллекции должны иметь одинаковую размерность и сравниваться по одной метрике. Именованные векторы позволяют хранить

несколько векторов в одной точке, каждый из которых может иметь собственные требования к размерности и метрике.

- Метрики расстояния используются для измерения сходства между векторами и должны быть выбраны при создании коллекции. Выбор метрики зависит от способа получения векторов и, в частности, от нейронной сети, которая будет использоваться для кодирования новых запросов.
- Точки являются центральной сущностью, с которой работает Qdrant, и они состоят из вектора и необязательного идентификатора и payload.
    - id: уникальный идентификатор векторов.
    - Вектор: многомерное представление данных, например, изображение, звук, документ, видео и т. д.
    - Payload: это объект JSON с дополнительными данными, которые можно добавить в вектор.
- Хранение: Qdrant может использовать один из двух вариантов хранения: хранилище **в оперативной памяти** (хранит все векторы в оперативной памяти, имеет самую высокую скорость, поскольку доступ к диску требуется только для сохранения) или хранилище **Memmap** (создает виртуальное адресное пространство, связанное с файлом на диске).

8.4 Графовая база данных PostgreSQL + Apache AGE

В качестве графовой базы данных используется PostgreSQL с расширением Apache AGE.

Apache AGE (A Graph Extension) — это расширение для PostgreSQL, которое добавляет функциональность графовой БД к стандартной реляционной СУБД. Оно реализует поддержку открытого стандарта Cypher Query Language, используемого в Neo4j, что позволяет выполнять сложные графовые запросы прямо внутри PostgreSQL. В контексте RAG-систем Apache AGE играет важную роль, обеспечивая работу со структурированными связями между данными, что дополняет традиционный векторный поиск и значительно расширяет возможности генерации точных и содержательных ответов.

Основное преимущество Apache AGE для ARAG заключается в возможности хранить и анализировать сложные взаимосвязи между сущностями, которые трудно или невозможно выразить через обычные векторные эмбеддинги.

Поскольку это расширение, а не отдельная СУБД, оно зависит от возможностей PostgreSQL и не включает некоторые продвинутые алгоритмы анализа графов, такие как центральность или кластеризация, без дополнительных настроек.

8.5 База данных для метаданных PostgreSQL

В качестве БД для хранения метаданных используется PostgreSQL.

Асинхронный клиент для взаимодействия с базой управляет жизненным циклом метаданных документов в PostgreSQL.

В базе выполняются следующие операции:

- Создаются новые записи о документах и записывается событие;
- Обновляется информация о существующем документе (например, его S3-путь или статус);
- Выполняется запрос информации о документе;
- Выполняется «мягкое удаление».