Инструкция по установке
Arenadata Advanced RAG (ARAG)

Москва
2025

Содержание:

1 Назначение документа

Настоящая инструкция определяет порядок развертывания и запуска программного обеспечения Arenadata Advanced RAG (ARAG) (Retrieval-Augmented Generation) с использованием Docker Compose в двух вариантах: с поддержкой CPU и GPU. Также инструкция регламентирует действия для работы в изолированных средах без доступа к сети Интернет.

2 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
Docker	Платформа для разработки, доставки и запуска приложений в контейнерах
Docker Compose	Инструмент для определения и запуска многоконтейнерных приложений Docker
GPU	Графический процессор, используется для ускорения вычислений моделей искусственного интеллекта
Hugging Face	Платформа, предоставляющая библиотеки и модели машинного обучения

3 Сокращения и обозначения

Сокращение	Наименование
API	(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
DDL	Data Definition Language
DML	Data Manipulation Language
IP	(Internet Protocol) – маршрутизируемый протокол сетевого уровня стека TCP/IP
MSSQL	Microsoft SQL Server
LDAP	(англ. – Lightweight Directory Access Protocol) – протокол прикладного уровня для доступа к службе каталогов
LLMs	Large language models
SSL	Secure Sockets Layer
SQL	(Structured Query Language) – декларативный язык программирования, применяемый для создания, модификации и управления данными в реляционных базах данных
UI	User interface
БД	База данных
ГБ	Гигабайт
ОС	Операционная система
ПО	Программное обеспечение
СУБД	Система управления базами данных
ТБ	Терабайт

4 Описание решения

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.

ARAG — это ПО, объединяющее поиск информации (retrieval) и генерацию текста (generation) для предоставления ответа, близкого по смыслу к запросу, даже если нет точного совпадения ключевых слов. ПО использует LLM-модели для извлечения сущностей и связей из фрагментов текста. Каждый чанк обрабатывается языковой моделью, которая извлекает из текста ключевые сущности (значимые объекты: персоны, организации, термины и т.д.) и отношения между ними.

5 Требования к узлу развертывания

Установка ARAG выполняется на хост под управлением Linux[1]. Для работы ARAG требуется наличие на хосте установленной и настроенной платформы запуска контейнерных приложений Docker или аналогичной, например, Podman. Текущему пользователю должны быть предоставлены привилегии запуска docker-контейнеров.

Системные требования для узла ARAG:

Ядро системы (Core Advanced RAG):

- CPU: 8+ ядер (Intel Xeon/AMD EPYC)
- RAM: 32+ ГБ
- Диск: 100+ ГБ SSD (для кэша и временных данных)
- Сеть: 1+ Гбит/с
- ОС: Linux (Ubuntu 22.04 LTS)
- Docker

Векторная база данных (Vector DB):

- CPU: 8+ ядер
- RAM: 64+ ГБ
- Диск: 1+ ТБ NVMe SSD
- Отдельная нода (рекомендуется)
- Docker

Графовая база данных (Graph DB):

- CPU: 8+ ядер
- RAM: 64+ ГБ
- Диск: 500+ ГБ SSD
- Отдельная нода (рекомендуется)

[1] Работа ПО на Windows возможна, но не описывается в данной инструкции.

- Docker

Требования к LLM серверу:

- GPU: 2+ x NVIDIA 6000Ada (48+ ГБ VRAM)
- / 2+ x NVIDIA 4090/3090 (24+ ГБ VRAM)
- CPU: 16+ ядер
- RAM: 64+ ГБ
- Диск: 500+ ГБ SSD
- Отдельный GPU-кластер (обязательно)
- Docker

Требования к Embedding-серверу:

- GPU: 1 x NVIDIA 4090/3090 (24+ ГБ VRAM)
- CPU: 8+ ядер
- RAM: 64+ ГБ
- Диск: 500+ ГБ SSD
- Отдельный GPU-кластер (не обязательно)
- Docker

Требования к RAGAS (Evaluation module):

- CPU: 4+ ядер
- RAM: 32+ ГБ
- Диск: 100+ ГБ (для тестовых датасетов)
- Docker

ОС: CentOS 7.9 / Ubuntu / Astra Linux;

Системное ПО: Docker / Docker CE / Podman.

Требования к сетевой инфраструктуре:

Внутренние подключения: сетевая связанность между узлами решения, отсутствие блокировок портов (TCP / UDP), производительность не ниже 1 Gb / sec.

6 Порядок развертывания

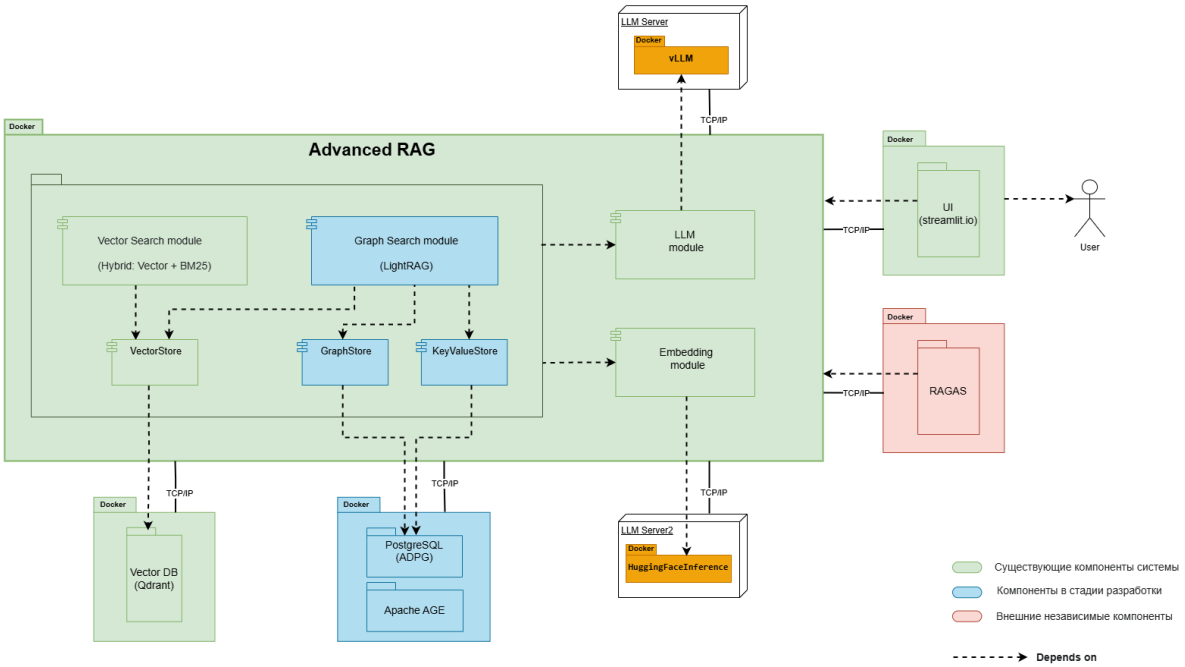

Рис.1 - Схема разделения по контейнерам

Установка с доступом в Интернет

1. Скопируйте все необходимые файлы проекта (включая `docker-compose.cpu.yml`, `docker-compose.gpu.yml` и др.) в директорию пользователя `advanced_rag`.
2. В зависимости от конфигурации системы выберите один из вариантов:

- Запуск на CPU

sudo docker-compose -f docker-compose.cpu.yml up

- Запуск на GPU

sudo docker-compose -f docker-compose.gpu.yml up

3. Для штатной остановки работы приложения выполните команду:

   sudo docker-compose -f docker-compose.<cpu/gpu>.yml down

4. Для аварийной остановки нажмите сочетание клавиш `Ctrl+C` в терминале, где запущено приложение.

Установка в изолированной среде (без доступа в Интернет):

1. Перенесите полученные `.tar`-файлы на целевую машину.
2. На целевой машине выполните команды для загрузки образов:

   docker load -i /<путь_к_файлу>/advanced_rag.tar

   docker load -i /<путь_к_файлу>/streamlit_ui.tar

   docker load -i /<путь_к_файлу>/qdrant.tar

Предварительная настройка кэша Hugging Face

Чтобы обеспечить работу в средах без доступа к сети Интернет, ПО настроено на использование локального кэша для моделей Hugging Face.

Каталог ./huggingface монтируется в advanced_rag контейнер по указанному пути /home/rag_user/.cache/huggingface. Это обеспечивает приложению доступ к загруженным моделям и кэшам.

Запуск приложения

Запустите приложение с использованием специальных compose-файлов, предназначенных для работы с предзагруженными образами:

- Запуск на CPU

sudo docker-compose -f docker-compose-images.cpu.yml up

- Запуск на GPU

  sudo docker-compose -f docker-compose-images.gpu.yml up

  Переменные среды:

  - HF_HUB_OFFLINE=1: Отключает попытки подключения к Hugging Face Hub для загрузки моделей, заставляя приложение использовать только локальный кэш.
  - TIKTOKEN_CACHE_DIR=/home/rag_user/.cache/huggingface/tokenizer: Дает указание tiktoken (библиотеке токенизации) использовать указанный каталог для хранения кэша токенизатора.
  - LLAMA_INDEX_CACHE_DIR=/home/rag_user/.cache/huggingface/hub: Указывает llama-index использовать указанный каталог для кэширования данных, связанных с индексом.
  - FASTEMBED_CACHE_PATH=/home/rag_user/.cache/huggingface/hub: Указывает fastembed использовать указанный каталог в качестве пути к кэшу.

  Проверка работоспособности API:

  Выполните команду:

  a) для cpu:

  model=<dense embedding model># например, sentence-transformers/all-MiniLM-L6-v2 docker run -p 8080:80 --pull always ghcr.io/huggingface/text-embeddings-inference:cpu-1.7 --model-id $model

  b) для gpu:

  `model=<dense embedding model>` # for example, `sentence-transformers/all-MiniLM-L6-v2`

`docker run --gpus all -p 8080:80 --pull always ghcr.io/huggingface/text-embeddings-inference:1.7 --model-id $model`

API для встраивания доступен по адресу: host:port/embed.

Вы можете проверить это с помощью команды ниже:

curl -X POST 'host:port/embed' \

--header 'Content-Type: application/json' \

--data '{"inputs": "test"}'

При установке создается три контейнера (см. рис.1):

- streamlit_ui
- qdrant
- advanced_rag

Адрес qdrant - http://<host>:6333/dashboard#/collections

Адрес UI - http://<host>:8501/

Для использования модели эмбединга, установленной внутри контейнера, надо выставить параметр local в config.yaml:

embeddings: dense_embeddings: selected_dense_embedding: "local"

После запуска всех трех контейнеров необходимо создать домены.

Пример запроса: curl --location ' http://127.0.0.1:8088/domains_load' \ --header 'Content-Type: application/json' \ --data '{"domains": [{"name": "ADB","description": "Arenadata DB (ADB) – это массивно-параллельная реляционная СУБД с открытым исходным кодом для хранилищ данных с гибкой горизонтальной масштабируемостью и колоночным хранением на основе PostgreSQL. Благодаря своим архитектурным особенностям и мощному оптимизатору запросов, ADB отличается особой надежностью и высокой скоростью обработки SQL-запросов над большими объемами

данных – поэтому Arenadata DB широко применяется для аналитики Big Data в промышленных масштабах. Для более удобной работы и построения практических задач любой сложности Arenadata DB поставляется вместе с рядом дополнительных инструментов, обеспечивающих интеграцию с внешними хранилищами данных, управление бинарными бэкапами и мониторинг запросов в режиме реального времени. Описанный функционал позволяет строить решения с полным покрытием всех процессов, связанных с сопровождением бизнес-систем."}, {"name": "ruRTD","description": "Датасет для тестирования русскоязычных RAG-систем."}]}'

Рекомендуется использовать Postman.

7 Настройка config manager

Для управления конфигурацией ПО используется config manager. Поддерживаются как локальные эмбеддинг модели, так и эмбеддинг по API.

1. Конфигурируются следующие параметры векторной БД:
    - local_persist_directory: Путь к данным Qdrant для локального (несерверного) режима.
    - url: URL-адрес сервера базы данных, например, http://qdrant:6333.
    - .collection_name: Имя коллекции.
    - batch_size: Размер пакета для загрузки векторов.
2. Конфигурирование используемых моделей:
    - model_name: Название модели.
    - base_url: Базовый URL-адрес сервера модели.

- token: токен доступа для модели.
- temperature: контролирует степень разнообразия реакций (от 0 до 1).
- presence_penalty: Штраф за использование одинаковых токенов.
- max_tokens: максимальное количество токенов, которые будут возвращены.
- timeout: время ожидания запроса в секундах.

3. Параметры Dense Embeddings:
   - selected_dense_embedding: указывается используемый бэкэнд. Должен соответствовать одному из ключей ниже dense_embeddings (например, local или text_embeddings_inference).
   - local: запускает модель локально в процессе приложения.
     - model_name: Имя модели для локальной загрузки, например, sentence-transformers/all-MiniLM-L6-v2.
     - device: Устройство для исполнения (cpu или cuda).
   - text_embeddings_inference: Использует удаленный сервер вывода на основе Hugging Face text-embeddings-inference.
     - model_name: Имя модели, настроенной на удаленном сервере, например, sentence-transformers/all-MiniLM-L6-v2.
     - base_url: URL-адрес работающего text-embeddings-inferencесервера, например, http://host.docker.internal:8080.

- timeout: время ожидания запроса в секундах.
- embed_batch_size: Максимальное количество текстов для встраивания в один запрос.

4. Параметры Sparse Embeddings:

- model_name: Модель для разреженных вложений. В настоящее время доступно только Qdrant/BM25

Обработка запроса пользователя

5. Настраиваются следующие параметры Chunking:
   - llm: LLM, используемый для обработки фрагментации текста. Доступные варианты: llm или small_llm.
   - chunk_size: максимальный размер одного текстового фрагмента.
   - chunk_overlap: Перекрытие между фрагментами текста.
   - selected_type: выбранный(е) тип(ы) интеллектуального разбиения на фрагменты, например, ["raptor", "densex"].
     - Если выбрано несколько типов (например, ["raptor", "densex"]), разбиение на фрагменты и извлечение будут выполняться с использованием этих методов.
     - Если выбран только один тип (например, ["raptor"]), разбиение на фрагменты и извлечение будут выполняться исключительно с использованием этого метода.
6. Параметры Raptor:
   - tree_depth: Глубина дерева кластеризации.

- threshold: Порог вероятности принадлежности к кластеру. Параметр threshold в функции GMM_cluster контролирует, насколько «уверенно» точка должна принадлежать кластеру. Увеличение threshold приведет к тому, что точки с меньшей вероятностью будут отнесены к какому-либо кластеру, что может сократить общее количество кластеров.
- max_length_in_cluster: Максимальная длина текста в кластере. Еще один параметр для управления количеством документов в кластере — на основе общей длины. Если кластер превышает эту длину, он разделяется.
- context_window: Окно контекста. Если параметр max_length_in_cluster превышает максимальное окно контекста LLM, то суммирование кластера будет выполняться рекурсивно (разделяя кластер на размеры context_window и суммируя каждый полученный фрагмент) - в конечном итоге приводя к суммированию сумм.
- num_workers: Количество потоков для обработки.
- summary_prompt: промт для кластера.

7. Параметры Densex:
    - num_workers: Количество потоков.
    - propositions_prompt: Промт для генерации предложений.

Пример конфигурации:

```
chunking:
llm: small_llm
chunk_size: 512
chunk_overlap: 50
selected_type: raptor
type:
```

```
  raptor:
    tree_depth: 3
    threshold: 0.1  #Параметр threshold в функции GMM_cluster контролирует,
    насколько "уверенно" точка должна принадлежать кластеру.
    Увеличение threshold приведет к тому, что точки с меньшей вероятностью будут
отнесены к какому-либо кластеру,
    что может уменьшить общее количество кластеров.
    max_length_in_cluster: 100000
    context_window: 30000
    num_workers: 10
    summary_prompt: >
      As a professional referee, write a concise and comprehensive summary of the
provided text in as much detail as possible.

  densex:
    num_workers: 5
    propositions_prompt: >
     #указывается промт
```

- Используемая модель указывается в конфигурационном файле, ее настройка - на стороне пользователя

Retriever / reranker/ Экстракторы

8. Настраиваемые параметры reranker:
   - top_n: Количество результатов для ранжирования. Конечное количество релевантных документов.
   - rerank_model: Модель для переоценки. В настоящее время доступна только colbert-ir/colbertv2.0.
   - score_threshold: Минимальный балл, который должен получить документ, чтобы считаться релевантным после переоценки. Документы с баллами ниже этого порога будут отброшены.
9. Настраиваемые параметры retriever:
   - similarity_top_k: контролирует конечное количество возвращаемых узлов.
   - sparse_top_k: показывает, сколько узлов будет извлечено из каждого плотного и разреженного запроса.

- retriever_weights: вес для объединения плотного и разреженного поиска. (Если установлено значение 0, будет выполняться только разреженный поиск. Если установлено значение 1, будет выполняться только плотный векторный поиск).

Например, если установлено sparse_top_k=5, то это означает, что будет извлечено 5 узлов с помощью разреженных векторов и 5 узлов с помощью плотных векторов. Параметр similarity_top_k управляет окончательным числом возвращаемых узлов. В приведенной выше настройке получается 10 узлов. Алгоритм слияния применяется для ранжирования и упорядочивания узлов из разных векторных пространств (в данном случае слияние с относительной оценкой). Значение параметра similarity_top_k=2 означает, что возвращаются два верхних узла после слияния.

10. Настраиваемые параметры экстракторов:

- llm: LLM, используемый для извлечения.
- num_workers: Количество потоков.

SummaryExtractor:

- enabled: логическое значение, указывающее, включен ли SummaryExtractor.
- summaries: Список, указывающий, какие типы краткого содержания следует включить. Возможные значения: «self», «prev» и «next».
    - self: Включить краткое содержание текущего фрагмента.

  - prev: Включить краткое содержание предыдущего фрагмента.
  - next: Включить резюме следующего фрагмента. Использование 'prev' и 'next' добавляет контекст к резюме текущего фрагмента. ВАЖНО: параметр SummaryExtractor предназначен для использования с большими фрагментами, где включение резюме смежных фрагментов обеспечивает ценный контекст. Это гарантирует, что большой фрагмент будет понят в более широком контексте окружающей информации.
- prompt_template: Шаблон для резюмирования каждого фрагмента.

KeywordExtractor:

- enabled: логическое значение, указывающее, включен ли KeywordExtractor.
- prompt_template: Шаблон для извлечения ключевых слов.

11. Настраиваемые параметры для больших контекстов:

Этот раздел управляет процессом формирования окончательного ответа, используя подход древовидного резюмирования для обработки потенциально больших контекстов.

- context_window: определяет размер контекстного окна для LLM. Если общий размер извлеченных соответствующих документов превышает это значение, контекст разбивается на несколько фрагментов, каждый из которых не больше context_window. Затем каждый фрагмент обрабатывается LLM асинхронно. Полученные резюме из каждого фрагмента затем объединяются и возвращаются в LLM для генерации окончательного, всеобъемлющего ответа. Это позволяет системе обрабатывать контексты, превышающие максимальное контекстное окно LLM. Например, если context_window 50 000 токенов, а извлеченный контекст составляет 150 000 токенов, контекст будет разделен на три фрагмента, каждый из которых обрабатывается независимо перед объединением для окончательного ответа.
- num_output: Устанавливает максимальное количество токенов, разрешенных в сгенерированном ответе.
- prompt: Шаблон для генерации ответа.