

Задача для Хакатона ИТ-Академии Samsung 2026

«Индексация и поиск по сообщениям»

Оглавление

Бизнес-задача

Требования и ограничения

Что требуется разработать

Что предоставляется на этапе проверки

Что предоставляется на этапе разработки

Требования к Index Service

Схема запроса POST /index

Схема ответа POST /index

Схема POST /sparse_embedding

Требования к Search Service

Схема запроса POST /search

Схема ответа POST /search

SLA на проверку

Разрешённые технологии

Процесс сдачи решения

Критерии оценки

Описание полей датасета

Базовое решение

Бизнес-задача

Более 20% пользователей корпоративного мессенджера VK WorkSpace ежедневно пользуются поиском. Для них в первую очередь важно качество поиска - точность и релевантность: насколько корректно находятся верные сообщения и насколько близко первый верный результат расположен к началу выдачи.

Чтобы поиск работал, необходимо:

1. предобработать и проиндексировать данные;
2. предоставить API для выполнения поисковых запросов к индексу.

Недавно мы выпустили обновлённый пайплайн поиска и улучшили качество его работы, но не уверены, что проверили все гипотезы по извлечению полезных для поиска метаданных из исходных сообщений и **предлагаем вам попробовать улучшить наши метрики, используя доступные нам технологии.**

Требования и ограничения

Что требуется разработать

- Сервис индексации (Index Service)
- Поисковый сервис (Search Service)

Что предоставляется на этапе проверки

- **Qdrant** - инстанс векторной СУБД, развёрнутый в тестирующей системе рядом с сервисами команды. Адрес и имя коллекции передаются через переменные окружения.
- **Внешний HTTP API инференса dense-эмбеддингов и rerank-модели.** URL-ы и API-ключи передаются через переменные окружения.

Ресурсы исполнения

- Каждый сервис запускается в изолированном окружении с лимитами 4 CPU / 7 GB RAM. Данные ресурсы могут использоваться под любую бизнес логику, при этом dense вектора вычисляются проверяющей системой вне этих ресурсов.

Что предоставляется на этапе разработки

- схема данных (в этом ТЗ и примерах датасета);
- пример реального чата и ground-truth вопросов (data/Go VK.json, набор test-вопросов);
- доступ к API инференса dense/rerank с ограничениями на частоту и объём запросов;
- шаблон решения (example-solution) с корректными контрактами и минимальным решением.

Требования к Index Service

- Поставляется Docker-образом.
- Должен реализовать эндпоинты:
 - GET **/health** - health-check, возвращает 200 OK;
 - POST **/index** - принимает порцию сообщений и возвращает чанки для индексации;
 - POST **/sparse_embedding** - принимает текст и возвращает sparse-векторы, совместимые с Qdrant.
- **Важно:** у контейнера нет доступа в интернет. Sparse-модель должна быть упакована внутрь образа.
- Адрес и порт прослушивания задаются переменными окружения HOST и PORT.

Схема запроса POST /index

```
type IndexAPIRequest struct {
    Data IndexAPIDataItem `json:"data"`
}

type IndexAPIDataItem struct {
    Chat      Chat    `json:"chat"`
    OverlapMessages []Message `json:"overlap_messages"`
    NewMessages []Message `json:"new_messages"`
}

type ChatType string

const (
    ChatTypePrivate ChatType = "private"
    ChatTypeGroup   ChatType = "group"
    ChatTypeChannel ChatType = "channel"
)

type Chat struct {
    ID      string    `json:"id"`
    Name    string    `json:"name"`
    SN      string    `json:"sn"`
    Type    ChatType  `json:"type"`
    IsPublic *bool     `json:"is_public"`
    MembersCount *int      `json:"members_count"`
    Members []map[string]any `json:"members"`
}
```

```

type Message struct {
    ID      string    `json:"id"`
    ThreadSN *string    `json:"thread_sn"`
    Time    int64      `json:"time"`
    Text    string    `json:"text"`
    SenderID string    `json:"sender_id"`
    FileSnippets string  `json:"file_snippets"`
    Parts   []map[string]any `json:"parts"`
    Mentions []string    `json:"mentions"`
    MemberEvent map[string]any `json:"member_event"`
    IsSystem bool      `json:"is_system"`
    IsHidden bool      `json:"is_hidden"`
    IsForward bool      `json:"is_forward"`
    IsQuote bool      `json:"is_quote"`
}

```

Схема ответа POST /index

```

type IndexAPIResponse struct {
    Results []IndexAPIResultItem `json:"results"`
}

```

```

type IndexAPIResultItem struct {
    PageContent string `json:"page_content"` // текст чанка, сохраняется в payload
    Qdrant
    DenseContent string `json:"dense_content"` // текст для построения dense-вектора
    SparseContent string `json:"sparse_content"` // текст для построения sparse-вектора
    MessageIDs []string `json:"message_ids"` // id сообщений, которые покрывает чанк
}

```

dense_content и sparse_content можно сделать равными page_content или формировать индивидуально - это точка для оптимизации качества.

Схема POST /sparse_embedding

```

type SparseEmbeddingRequest struct {
    Texts []string `json:"texts"`
}

```

```

type SparseVector struct {
    Indices []int `json:"indices"`
}

```

```
    Values []float64 `json:"values"`  
}
```

```
type SparseEmbeddingResponse struct {  
    Vectors []SparseVector `json:"vectors"`  
}
```

Этот эндпоинт вызывается тестирующей системой как на этапе индексации, так и на этапе поиска (через прокси для Search Service).

Требования к Search Service

- Поставляется Docker-образом.
- Должен реализовать эндпоинты:
 - GET **/health** - health-check, возвращает 200 OK;
 - POST **/search** - принимает обогащённый вопрос, возвращает ранжированный список `message_ids`.
- **Важно:** у контейнера нет доступа в интернет. Обращения к dense-модели, sparse-модели и rerank-модели идут через проверяющую систему.
- Переменные окружения, передаваемые сервису:
 - HOST, PORT - адрес и порт прослушивания;
 - API_KEY - per-task JWT для Qdrant и inference-эндпоинтов;
 - EMBEDDINGS_DENSE_URL - URL dense-эмбедингов;
 - RERANKER_URL - URL rerank-модели;
 - QDRANT_URL, QDRANT_COLLECTION_NAME - адрес Qdrant и имя тестовой коллекции;
 - QDRANT_DENSE_VECTOR_NAME, QDRANT_SPARSE_VECTOR_NAME - имена полей векторов.

Схема запроса POST /search

```
type DateRange struct {  
    From string `json:"from"`  
    To   string `json:"to"`  
}
```

```
type Entities struct {  
    People   []string `json:"people"`  
    Emails   []string `json:"emails"`  
    Documents []string `json:"documents"`  
    Names    []string `json:"names"`  
    Links    []string `json:"links"`  
}
```

```
type Question struct {  
    Text      string `json:"text"`  
    Asker     string `json:"asker"`  
    AskedOn   string `json:"asked_on"`  
    Variants  []string `json:"variants"`  
    Hyde      []string `json:"hyde"`  
    Keywords  []string `json:"keywords"`  
}
```

```
    Entities    Entities `json:"entities"`
    DateMentions []string `json:"date_mentions"`
    DateRange    *DateRange `json:"date_range"`
    SearchText    string    `json:"search_text"`
}
```

```
type SearchAPIRequest struct {
    Question Question `json:"question"`
}
```

Схема ответа POST /search

```
type SearchAPIResultItem struct {
    MessageIDs []string `json:"message_ids"`
}
```

```
type SearchAPIResponse struct {
    Results []SearchAPIResultItem `json:"results"`
}
```

Результаты должны быть отсортированы по убыванию релевантности.

SLA на проверку

- Индексация всего тестового датасета - ориентировочно 15 минут.
- Поиск ответа на 1 вопрос - не более 60 секунд (наш запрос в ваш Search Service).
- Полный прогон проверки решения ограничен лимитом, указанным в личном кабинете команды.

Разрешённые технологии

- Любой язык программирования, упаковываемый в Docker-образ (linux/amd64).
- Любые open-source библиотеки для chunking, sparse-эмбедингов, retrieval и rerank.
- Для dense-эмбедингов и rerank - только предоставленный внешний HTTP API проверяющей системы (dense-модель Qwen/Qwen3-Embedding-0.6B, rerank-модель nvidia/llama-nemotron-rerank-1b-v2).
- Для хранения векторов - только предоставленный инстанс Qdrant.

Запрещено:

- обращаться в интернет из контейнеров index и search;
- менять контракты эндпоинтов /index, /sparse_embedding, /search;
- использовать сторонние LLM/эмбединг-API во время проверки.

Процесс сдачи решения

1. В личном кабинете хакатона получить логин, пароль и team_id для Docker registry.
2. Добавить registry проверяющей системы в insecure-registries Docker-демона.
3. Аутентифицироваться:
`docker login <registry> -u <login> -p <password>`
4. Собрать образы под linux/amd64 с тегами строгого формата:
 - <registry>/<team_id>/index-service:latest
 - <registry>/<team_id>/search-service:latest
5. Запустить образы в registry.
6. На странице оценивания нажать «Запустить оценивание».
7. Дождаться, пока задача пройдет статусы creating → starting → preparing → indexing → evaluating → done.
8. Получить итоговые метрики и увидеть результат в таблице лидеров.

Критерии оценки

Оценка производится по двум метрикам, усреднённым по всем вопросам датасета:

- **Recall@K** — доля вопросов, для которых хотя бы один из K найденных чанков соответствует эталонному сообщению.
- **nDCG@K** — нормализованный discounted cumulative gain на первых K результатах; учитывает позицию релевантных сообщений в выдаче.

В нашем случае K = 50. Если ваше решение возвращает больше 50-и `message\_id`, то все после первых 50-и отбрасываются и не учитываются.

Итоговый score:

$$\text{score} = \text{recall_avg} * 0.8 + \text{ndcg_avg} * 0.2$$

Описание полей датасета

- id - уникальный идентификатор записи из исходного dataset.jsonl. Обеспечивает стабильную связь между исходным и преобразованным датасетом.
- question - объект со всей информацией, помогающей искать релевантные сообщения по пользовательскому запросу.
 - question.text - оригинальный текст вопроса без нормализации. Основной запрос, от которого строятся все дополнительные представления.
 - question.asker - идентификатор автора вопроса (обычно email). Полезен как дополнительный фильтр и как сигнал для поиска сообщений, где фигурирует этот человек.
 - question.asked_on - дата вопроса в формате ISO YYYY-MM-DD (например, 2025-10-20). Нужна для интерпретации относительных дат: «сегодня», «13 октября», «вчера».
 - question.variants - массив переформулировок исходного вопроса: краткие, разговорные, нормализованные и keyword-heavy версии.
 - question.hyde - массив гипотетических текстов в стиле HyDE, которые могли бы встретиться в релевантных сообщениях. Не фактические сообщения, а правдоподобные формулировки, помогающие retrieval.
 - question.keywords - массив ключевых слов и фраз, извлечённых из вопроса. Подходит для keyword-search, фильтрации и дополнительного индексирования.
 - question.entities - структурированные сущности, найденные в вопросе:
 - people - упомянутые люди;
 - emails - email-адреса, включая адреса людей из people;
 - documents - документы, статьи, страницы, приказы, заявления;
 - names - продукты, сервисы, системы, команды, внутренние названия;
 - links - ссылки из текста вопроса или выведенные при его нормализации.
 - question.date_mentions - массив всех временных упоминаний: абсолютные даты, относительные даты, интервалы. Например: ["13 октября", "сейчас", "20 октября в 19:00"].
 - question.date_range - нормализованный диапазон дат, если вопрос задаёт интервал; иначе null. Формат: {"from": "2025-10-13T00:00:00Z", "to": "2025-10-20T12:30:00Z"}. Даты нормализуются относительно question.asked_on.
 - question.search_text - версия вопроса, оптимизированная для полнотекстового и векторного поиска.
- answer - эталонная информация об ответе, нужная для связи запроса с правильными сообщениями.
 - answer.text - ожидаемый правильный ответ из исходного датасета без изменений: ссылка, имя, дата, краткая формулировка или иной целевой ответ.
 - answer.message_ids - массив id сообщений, в которых содержится или подтверждается правильный ответ. Даже если в исходнике id переданы одной строкой через переносы, здесь ожидается именно список строк.
- metadata - метаданные вопроса:
 - metadata.team - команда сотрудника (по умолчанию CUTE);

- metadata.response_type - тип ответа;
- metadata.retrieve_type - тип поиска.

Базовое решение

Шаблон доступен в репозитории

<https://github.com/vktechdev/it-academy-hackathon-solution-example>. Он содержит:

- index/ - минимальный Python/FastAPI-сервис индексации с примерным chunking и локальной sparse-моделью (Qdrant/bm25 через fastembed);
- search/ - минимальный Python/FastAPI-сервис поиска с dense+sparse prefetch и rerank через внешние API проверяющей системы;
- docker-compose.yml - локальный запуск qdrant, index и search;
- data/Go VK.json - пример реального чата для отладки формата входных данных.

Это не эталон по качеству поиска, а **корректный контракт и отправная точка**.

Разрешено и рекомендуется менять: логику chunking, формирование dense_content и sparse_content, retrieval- и rerank-pipeline, любые эвристики и фильтры.

Запрещено менять сигнатуры POST /index, POST /sparse_embedding, POST /search.