МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»
Кафедра математической кибернетики и компьютерных наук

РАЗРАБОТКА СИСТЕМЫ МУЛЬТИМОДАЛЬНОГО СЕМАНТИЧЕСКОГО ПОИСКА ПО ДАННЫМ КОРПОРАТИВНЫХ МЕССЕНДЖЕРОВ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Лезгян Андрея Саркисовича

Научный руководитель
зав. каф. техн. пр.,
доцент, к. ф.-м. н. _____ И. А. Батраева

Заведующий кафедрой
доцент, к. ф.-м. н. _____ С. В. Миронов

Саратов 2026

1 Аналитический обзор предметной области и существующих подходов

1.1 Полнотекстовый и векторный поиск

Задача полнотекстового поиска состоит в нахождении документов, содержащих слова или фразы из пользовательского запроса, и их ранжирования по степени релевантности. Для этого текст предварительно индексируется: разбивается на токены, нормализуется и сохраняется в специальной структуре данных. Одним из распространенных подходов к ранжированию является BM25. Данный алгоритм учитывает частоту терминов в документе, их распространенность в коллекции и длину документа. Такой поиск хорошо подходит для точных совпадений: технических терминов, названий модулей, идентификаторов задач, фрагментов ошибок, имен файлов и команд запуска. Основное ограничение связано с зависимостью от лексического совпадения между запросом и документом. Однако часто релевантный фрагмент может не содержать слов из запроса, хотя совпадать с ним по семантике.

Векторный, или семантический, поиск основан на представлении текстов, изображений и иных объектов в виде числовых векторов фиксированной размерности. Такие embedding-представления кодируют не только поверхностные лексические признаки, но и смысловые свойства объекта: близкие по содержанию сообщения располагаются в векторном пространстве на малом расстоянии друг от друга. На этапе индексации каждая поисковая единица преобразуется в embedding-вектор и сохраняется в поисковом индексе. На этапе выполнения запроса пользовательская фраза кодируется той же моделью, после чего система осуществляет поиск ближайших объектов. Однако векторный поиск может быть менее точным для редких технических идентификаторов, кодов ошибок, имен файлов и команд, поскольку такие элементы не всегда хорошо отражаются в смысловом пространстве. Кроме того, построение векторов обычно требует больше вычислительных ресурсов, чем классический полнотекстовый поиск.

1.2 Мультимодальный поиск и обработка вложений

Мультимодальный поиск расширяет классический и семантический поиск за счет обработки данных разных типов. В корпоративном мессенджере информация содержится не только в текстовых сообщениях, но и во вложениях: документах, таблицах, логах, изображениях, скриншотах и аудиофайлах. Ес-

ли индексировать только текст сообщений, значительная часть корпоративного знания останется недоступной.

Текстовые документы, такие как PDF, DOCX, TXT, Markdown, CSV, JSON, LOG и табличные файлы, могут содержать инструкции, технические отчеты, выгрузки данных и фрагменты документации. Для их обработки применяется извлечение текста, после чего он нормализуется, разбивается на фрагменты и индексируется аналогично обычным сообщениям. Изображения и скриншоты требуют отдельного этапа обработки, поскольку их содержание изначально не представлено в виде машинно-читаемого текста. Для решения этой задачи используется OCR — оптическое распознавание символов. Оно преобразует изображение в текстовое представление, которое затем может быть включено в поисковый индекс. Для аудиофайлов применяется ASR — автоматическое распознавание речи. Система преобразует аудиозапись в текстовый транскрипт, после чего он проходит тот же конвейер обработки: нормализацию, чанкинг, построение embedding-представления и индексацию.

Главная задача мультимодальной обработки состоит в формировании единой поисковой единицы. Независимо от того, был ли исходный объект сообщением, документом, скриншотом или аудиозаписью, после обработки он должен содержать текстовое представление, embedding-вектор, тип источника, модальность и метаданные. Такой подход упрощает дальнейшую индексацию и поиск, поскольку поисковый модуль работает с единым набором объектов.

Для корпоративной системы также важна устойчивость обработки. Ошибка обработки одного файла не должна останавливать весь процесс индексации, поэтому архитектура должна предусматривать сохранение исходных данных, статусы обработки и возможность повторного запуска.

1.3 Гибридное ранжирование и оценка качества поиска

Гибридный поиск объединяет несколько каналов извлечения результатов: полнотекстовый и векторный поиски.

Ключевая проблема гибридного ранжирования связана с объединением результатов, полученных различными моделями. Оценки BM25 и cosine similarity имеют различную природу и находятся в разных шкалах, поэтому их непосредственное сложение или сравнение может приводить к нестабильным результатам. Для устранения этой проблемы целесообразно использовать методы позднего объединения, основанные не на абсолютных значениях score, а на

позициях документов в ранжированных списках.

Одним из таких методов является Reciprocal Rank Fusion (RRF). Данный метод присваивает документу итоговую оценку на основании его рангов в нескольких списках выдачи: чем выше объект расположен в каждом из списков, тем больший вклад он получает в итоговый результат.

В упрощенном виде оценка RRF может быть записана следующим образом:

$$\text{RRF}(d) = \sum_{i=1}^{m} \frac{1}{k + \text{rank}_i(d)},$$

где m — число объединяемых списков, $\text{rank}_i(d)$ — позиция документа в i-м списке, k — сглаживающий параметр. Такой подход не требует нормализации BM25-score и cosine similarity, что делает его удобным для практических гибридных систем.

Качество поисковой системы оценивается с помощью метрик информационного поиска. Precision@k показывает долю релевантных документов среди первых k результатов, Recall@k отражает долю найденных релевантных документов, MRR@k учитывает позицию первого релевантного результата, а nDCG@k оценивает качество ранжирования с учетом градаций релевантности.

Для корпоративного поиска особенно важны метрики при малых значениях k, например @5, поскольку пользователь мессенджера, как правило, ожидает увидеть требуемый фрагмент в верхней части выдачи и редко просматривает большое число результатов. Дополнительно необходимо измерять среднее время ответа, поскольку даже качественная выдача теряет ценность при большой задержке.

Таким образом, гибридная архитектура является наиболее оптимальной для поиска по данным корпоративных мессенджеров. Она учитывает лексические совпадения, смысловую близость и содержимое вложений, что делает ее более подходящей для реальной структуры корпоративных данных, чем изолированный полнотекстовый или только векторный поиск.

2 Программная реализация

2.1 Архитектура программной системы

Архитектура построена по многослойному принципу и разделена на два основных контура: контур индексации данных и контур пользовательского поиска. Такое разделение соответствует разной природе процессов. Индексация включает длительную обработку сообщений и вложений, построение векторов и синхронизацию с поисковым индексом. Пользовательский поиск, напротив, должен выполняться быстро и работать уже с подготовленными объектами.

В контуре индексации внешним источником данных является корпоративный мессенджер Mattermost. Пользователи отправляют сообщения, прикрепляют файлы, отвечают в тредах, а система получает эти данные через REST API и webhook-обработчик. Слой MessengerAdapter преобразует исходные JSON-объекты Mattermost во внутренний формат, благодаря чему последующие компоненты системы работают с унифицированными объектами и не зависят от деталей внешнего API.

После получения события FastAPI-сервис передаёт сообщение и сведения о вложениях в IngestionService. На данном этапе исходное сообщение и сведения о вложениях сохраняется в PostgreSQL. Это важное архитектурное решение: система сначала сохраняет факт получения объекта, а затем выполняет ресурсоемкую обработку. Если OCR, ASR или разбор документа завершатся ошибкой, исходные данные не будут потеряны, а обработка может быть выполнена повторно.

Результатом обработки являются объекты SearchItem — единые поисковые единицы, предназначенные для загрузки в Elasticsearch. Каждый SearchItem содержит текстовое представление объекта, embedding-вектор, модальность, тип источника, идентификаторы сообщения, канала и файла, а также дополнительные метаданные, необходимые для фильтрации и восстановления исходного контекста. Далее ElasticSyncService выбирает из PostgreSQL подготовленные элементы со статусом pending и синхронизирует их с поисковым индексом.

2.2 Интеграция с Mattermost и модель данных

Интеграционный слой реализован через общий интерфейс MessengerAdapter. Он задаёт набор операций, необходимых для работы с корпоративным мессенджером: получение списка каналов, чтение сообщений канала, получение

сведений о файле, скачивание вложения, отправку ответа пользователю и разбор входящего webhook-payload. Такая абстракция отделяет бизнес-логику системы от особенностей Mattermost API и делает основную логику независимой от конкретной платформы корпоративной коммуникации.

Адаптер Mattermost использует REST API и токен бота, передаваемый в HTTP-заголовке Authorization. Получение сообщений выполняется постранично, поскольку история канала может насчитывать огромное количество сообщений. Каждый пост преобразуется во внутреннюю модель, в которой сохраняются идентификаторы канала и сообщения, текст, сведения о треде, дата создания и список файлов. Скачивание вложений также находится внутри адаптера, поэтому модули OCR, ASR и извлечения текста работают уже с локальными файлами и не знают, из какого мессенджера они были получены.

Модель данных разделяет исходные и подготовленные объекты. RawMessage и RawFile используются для хранения того, что было получено из Mattermost. Они нужны для воспроизводимости обработки и повторной индексации. SearchItem представляет уже нормализованную поисковую единицу. В ней текст сообщения, фрагмент документа, OCR-результат изображения или ASR-транскрипт аудио приводятся к единой структуре.

Ключевое преимущество такой модели состоит в единообразии дальнейшей обработки. После этапа подготовки поисковому модулю уже не важно, был ли объект обычным сообщением, PDF-файлом, скриншотом или аудиозаписью. Все они имеют текстовое поле, embedding-вектор, модальность и метаданные. Это снижает связанность компонентов и упрощает добавление новых типов вложений.

2.3 Обработка сообщений и вложений

Конвейер обработки начинается с нормализации текста. Система удаляет нулевые символы, приводит переводы строк к единому виду, сокращает повторяющиеся пробелы и ограничивает количество пустых строк. Это необходимо, поскольку сообщения и файлы поступают из разных источников и могут содержать технический мусор, нестандартные переносы строк или фрагменты разметки.

Длинный текст разбивается на чанки. В работе выбраны настраиваемые параметры TEXT_CHUNK_SIZE и TEXT_CHUNK_OVERLAP, по умолчанию 1200 и 200 символов соответственно. Перекрытие между фрагментами позво-

ляет сохранять контекст на границах чанков: если релевантная информация находится в конце одного фрагмента и продолжается в начале следующего, она не теряется при индексировании.

Для документов реализована поддержка PDF, DOCX, XLSX, XLSM, TXT, MD, CSV, JSON и LOG. PDF обрабатывается с помощью pypdf, документы DOCX — через python-docx, таблицы Excel — через openpyxl. Для текстовых файлов предусмотрена работа с несколькими кодировками: UTF-8, CP1251 и Latin-1. Это важно для корпоративных архивов, где старые логи и экспортированные файлы могут быть сохранены в разных форматах.

Если вложение является изображением или скриншотом, для него выполняется OCR. После чего извлеченный с текст проходит тот же путь, что и обычное сообщение: нормализацию, чанкинг, построение embedding-представления и сохранение в PostgreSQL. Это позволяет искать по тексту интерфейсных ошибок, схем, фотографий документов и подписей на изображениях. Отдельное место в системе занимает обработка аудиофайлов. Для них используется ASR-блок на основе Whisper. Аудиосообщение преобразуется в текстовый транскрипт, после чего этот текст включается в общий поисковый конвейер. За счет этого голосовые сообщения становятся равноправными объектами поиска: пользователь может найти обсуждение, даже если оно было произнесено, а не написано в чате.

2.4 Мультимодальная обработка и embedding-представления

Построение векторных представлений вынесено в модуль EmbeddingHub. Он скрывает детали загрузки и вызова моделей и предоставляет единый интерфейс для кодирования текстовых документов, пользовательских запросов и изображений. Такой слой упрощает замену моделей: при изменении embedding-модели не требуется переписывать IngestionService, поисковый модуль или адаптер мессенджера.

В реализации используется нормализация embedding-векторов. Это делает косинусную близость более устойчивой и согласуется с настройкой dense_vector-поля в Elasticsearch, где применяется cosine similarity. Для текстовых фрагментов строятся векторы документов, для пользовательского запроса — отдельный вектор запроса, а для изображений может формироваться визуальное embedding-представление.

Практический смысл embedding-подхода состоит в том, что система полу-

чает возможность искать по смыслу, а не только по совпадению слов. Например, запрос пользователя может описывать проблему другими словами, чем те, которые использовались в исходной переписке. Векторное представление повышает вероятность нахождения таких фрагментов, а гибридная схема дополняет его точными совпадениями и полнотекстовым ранжированием.

2.5 PostgreSQL как хранилище и журнал обработки

В разработанной архитектуре PostgreSQL используется для хранения исходных данных, буферизации результатов обработки и ведении журнала состояний подготовленных объектов. Прямое обновление Elasticsearch могло бы упростить архитектуру, однако при временной недоступности поискового ядра возникал бы риск потери новых сообщений или вложений. Использование транзакционного хранилища устраняет этот риск, поскольку каждая поисковая единица предварительно фиксируется в базе данных.

Для хранения данных используются три основные таблицы: raw_messages, raw_files и search_items. Первые две таблицы содержат исходные сообщения и сведения о вложениях, полученные из мессенджера. Таблица search_items хранит результаты обработки: текстовые фрагменты, embedding-векторы, модальность, тип источника и метаданные. Такое разделение позволяет повторно обрабатывать старые сообщения при изменении параметров чанкинга, OCR, ASR или embedding-модели.

Состояние обработки отражается отдельными статусами. В raw_messages и raw_files используется processed_status: новые записи имеют статус pending, успешно обработанные — processed, а проблемные — failed. В search_items поле es_status отвечает уже за синхронизацию с Elasticsearch: pending означает ожидание отправки, indexed — успешную загрузку, failed — ошибку синхронизации.

Для сохранения применяется механизм upsert. Если сообщение или файл поступает впервые, запись создается. Если объект уже есть в базе, обновляются его содержимое и атрибуты. Аналогично обновляются SearchItem: при изменении данных поисковая единица снова переводится в pending и будет переотправлена в Elasticsearch при следующей синхронизации.

2.6 Синхронизация с Elasticsearch

Elasticsearch выполняет роль основного поискового ядра системы. В него попадают не сырые сообщения и файлы, а уже подготовленные SearchItem. Это принципиально поскольку поисковый индекс содержит унифицированные объекты, пригодные для полнотекстового и векторного поиска, а PostgreSQL остается источником исходных данных и состояния обработки.

Синхронизация вынесена в отдельный сервис ElasticSyncService. Он периодически выбирает из PostgreSQL записи со статусом pending, проверяет наличие индекса, при необходимости создает его и выполняет bulk-загрузку документов. После успешной отправки записи помечаются как indexed. При ошибке отдельные элементы могут получить статус failed, не блокируя обработку остальных объектов.

Периодический запуск синхронизации реализован через APScheduler. Интервал синхронизации задается параметром ES_SYNC_INTERVAL_SECONDS и по умолчанию составляет 600 секунд. В результате индекс обновляется в режиме, близком к реальному времени. Новые сообщения становятся доступными для поиска не мгновенно, но достаточно быстро для типичных корпоративных сценариев. При необходимости пользователь может инициировать отправку подготовленных объектов вручную с помощью команды `/kb sync`.

Mapping индекса Elasticsearch включает служебные поля item_id, messenger, source_type, modality, channel_id и created_at, объект metadata, текстовое поле text и dense_vector-поле embedding. Текст индексируется с анализатором для русскоязычных фрагментов, а embedding хранится как векторное поле с косинусной близостью. Это позволяет объединить фильтрацию по метаданным, полнотекстовый поиск и поиск по смысловой близости.

2.7 Гибридный поиск

Пользовательский поиск выполняется через SearchEngine. Когда пользователь отправляет запрос, поисковый модуль строит embedding запроса и выполняет два независимых обращения к Elasticsearch. Первое обращение — лексическое, оно использует полнотекстовый индекс, фразовые совпадения, совпадение всех терминов и fuzzy-поиск. Второе обращение — векторное, оно выполняет kNN-поиск по dense_vector-полю.

Лексический канал хорошо работает для точных технических терминов,

имен файлов, названий библиотек, кодов ошибок и устойчивых словосочетаний. Векторный канал лучше справляется с ситуациями, когда пользователь формулирует запрос естественным языком и не повторяет слова из исходного сообщения. Вместе они покрывают больше сценариев, чем каждый из подходов отдельно.

Объединение результатов выполняется методом Reciprocal Rank Fusion. Его преимущество в том, что он работает с позициями документов в отдельных списках и не требует приводить BM25-score и cosine similarity к общей шкале. Если объект высоко ранжирован и в лексическом, и в векторном поиске, его итоговая оценка растет. Если объект найден только одним каналом, он все равно может попасть в выдачу, но будет конкурировать с результатами другого канала.

Поддерживается ограничение поиска по текущему каналу мессенджера. Для команды `/kb search here` к запросу добавляется фильтр по channel_id. Это полезно, когда пользователь помнит контекст обсуждения и хочет искать только в конкретном проектном или инцидентном канале. Таким образом, система сочетает общий поиск по базе знаний и локальный поиск по текущему рабочему пространству.

2.8 Интерфейс пользователя в Mattermost

Пользователь взаимодействует с системой через slash-команду `/kb`. Это решение удобно для корпоративного поиска, потому что не требует отдельного веб-интерфейса: пользователь вводит команду в привычном канале и получает ответ там же. Для служебных команд и персональной выдачи используется ephemeral-ответ, который не засоряет общий канал и виден только инициатору запроса.

Команда `/kb help` выводит справку. Команда `/kb index` запускает фоновое чтение истории доступных каналов, сохраняет сырые сообщения и сведения о вложениях в PostgreSQL, а затем запускает их обработку. Команда `/kb sync` немедленно отправляет подготовленные SearchItem со статусом pending в Elasticsearch. Команда `/kb search` выполняет гибридный поиск по всему доступному индексу, а `/kb search here` ограничивает его текущим каналом.

Обработчик slash-команд реализован как FastAPI endpoint `/command`. Он проверяет токен команды, разбирает пользовательский ввод, выбирает нужное действие и возвращает ответ в формате, ожидаемом Mattermost. Длительные

операции, например индексация истории каналов, запускаются в фоне, чтобы HTTP-запрос Mattermost не ожидал завершения всей обработки.

Результаты поиска форматируются как список найденных фрагментов. Для каждого результата показываются тип источника, модальность, идентификаторы канала, сообщения и файла, а также короткий preview текста. Если найденный объект связан с вложением, дополнительно выводится имя файла из метаданных. Это помогает пользователю понять, найдено ли обычное сообщение, фрагмент документа, OCR-текст изображения или ASR-транскрипт аудио.

2.9 Экспериментальная проверка

Практическая проверка системы проводилась на нескольких уровнях. На первом уровне проверялись отдельные функции: нормализация текста, чанкинг, определение расширений файлов, OCR, ASR и форматирование выдачи. На втором уровне тестировалась интеграция с PostgreSQL: создание таблиц, сохранение сырых сообщений и файлов, повторный upsert, выборка записей со статусом pending, формирование SearchItem и подготовка объектов к синхронизации с Elasticsearch.

Также проверялись функции поискового ядра: создание индекса, bulk-загрузка документов, лексический поиск, kNN-поиск и RRF-объединение результатов. Отдельный уровень проверки составляли end-to-end сценарии через Mattermost, где пользователь отправляет команду, система обрабатывает запрос, обращается к индексу и возвращает отформатированную выдачу в мессенджер.

Для экспериментальной оценки использовался открытый датасет freeCodeCamp Gitter Chat за 2015–2017 годы. Он содержит сообщения из публичного чатрума и по структуре близок к данным корпоративного мессенджера. Поскольку исходные сообщения были на английском языке, перед экспериментом они были переведены на русский. Это приблизило данные к целевому сценарию работы системы и позволило оценивать поиск по русскоязычным запросам.

Тестовая выборка включала 40 поисковых запросов. Они отражали типичные сценарии: поиск по точным техническим терминам, поиск по смысловому описанию проблемы и поиск информации, находящейся во вложениях. Для каждого запроса формировался список релевантных сообщений, документов или мультимодальных объектов. Затем одна и та же выборка проверялась в режимах BM25, dense retrieval, гибридного поиска с RRF и гибридного поиска с учетом

OCR/ASR-данных.

Для оценки использовались метрики Precision@5, Recall@5, MRR@5 и nDCG@5. Precision@5 показывает долю релевантных документов среди первых пяти результатов, Recall@5 отражает полноту нахождения релевантных объектов, MRR@5 учитывает позицию первого релевантного результата, а nDCG@5 позволяет оценивать качество ранжирования с учетом градаций релевантности. Дополнительно фиксировалось среднее время ответа системы.

Полнотекстовый поиск показал следующие результаты: Precision@5 = 0,54, Recall@5 = 0,32, MRR@5 = 0,61 и nDCG@5 = 0,56 при среднем времени ответа 140 мс. Векторный поиск улучшил качество по всем основным метрикам: Precision@5 составил 0,59, Recall@5 — 0,40, MRR@5 — 0,68, nDCG@5 — 0,61. Однако время ответа выросло до 1350 мс, что связано с высокой вычислительной сложностью построения векторов.

Гибридный поиск дал лучший результат: Precision@5 = 0,68, Recall@5 = 0,46, MRR@5 = 0,74 и nDCG@5 = 0,73. Среднее время ответа составило 1620 мс. Оно выше, чем у изолированных baseline-подходов, но остается приемлемым для интерактивного поиска в корпоративном мессенджере. Рост времени является ожидаемой платой за объединение нескольких каналов извлечения и последующего ранжирования.

Результаты эксперимента подтверждают целесообразность гибридной схемы. BM25 обеспечивает быстрые и точные совпадения по терминам, векторный поиск находит смысловые совпадения, а RRF позволяет устойчиво объединить результаты. Особенно важным является вклад обработки вложений: OCR и ASR расширяют область поиска за пределы обычных текстовых сообщений.