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 МОДЕЛИ GRAPHRAG ДЛЯ АНАЛИЗА РУССКОЯЗЫЧНЫХ ТЕКСТОВ

АВТОРЕФЕРАТ БАКАЛАВРСКОЙ РАБОТЫ

студентки 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Стюхиной Ангелины Сергеевны

Научный руководитель
доцент, к. ф.-м. н. _____ М. И. Сафрончик

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

Саратов 2026

ВВЕДЕНИЕ

Актуальность темы. Современное развитие искусственного интеллекта (ИИ) привело к внедрению больших языковых моделей (англ. Large Language Model, LLM) в повседневную жизнь людей. Уже сейчас пользователи привыкают доверять краткой выжимке от искусственного интеллекта вместо того, чтобы проверять информацию в нескольких источниках.

Хотя LLM понимают и могут генерировать естественный язык, они ограничены их собственной архитектурой. Среди таких ограничений можно выделить «галлюцинации», то есть воспроизведение правдоподобной, но искаженной информации, низкую точность при работе с локальными или узко специализированными данными, размер контекста, который способна «помнить» модель, и другие. При этом, непрерывное дообучение моделей для обхода этих ограничений нецелесообразно из-за сложности и стоимости процесса.

Одним из наиболее распространённых подходов, направленных на решение проблемы «галлюцинаций», стала генерация с дополненной выборкой (англ. Retrieval-Augmented Generation, RAG). RAG комбинирует генеративные возможности языковой модели с поиском данных во внешних хранилищах, что позволяет значительно повысить качество и обоснованность ответов.

Традиционные RAG-системы опираются в основном на векторный поиск, который учитывает семантическую близость текстов, но не отражает структурные связи между сущностями предметной области. Это снижает качество работы со структурированной информацией, что особенно видно при ответах на так называемые multi-hop вопросы, то есть вопросы, требующие для ответа информацию из нескольких источников.

Следующим шагом развития RAG стала генерация с дополненной выборкой, основанная на графах (англ. Graph-Based Retrieval-Augmented Generation, GraphRAG), которая объединила векторный поиск с графами знаний. Граф знаний — это структура данных, содержащая узлы и связи между ними для представления сущностей предметной области. Граф знаний способен интегрировать структурированные и неструктурированные данные для обеспечения целостного представления информации.

В рамках бакалаврской работы рассматривается разработка и улучшение русскоязычной системы GraphRAG, предназначенной для анализа пользовательских документов и ответов на вопросы по этим документам. Особое вни-

мание уделяется сравнению GraphRAG и RAG моделей и созданию веб-клиента для использования системы.

Целью бакалаврской работы является разработка диалоговой системы на основе метода GraphRAG для поиска информации в пользовательских документах на русском языке, обеспечивающей высокую точность ответов, снижение в них «галлюцинаций» и удобство использования.

Для достижения указанной цели необходимо решить следующие задачи.

1. Подробно рассмотреть природу ограничений LLM.
2. Проанализировать и сравнить подходы RAG и GraphRAG.
3. Реализовать классическую RAG-систему.
4. Изучить современные решения для построения графов знаний и реализовать их в GraphRAG-системе.
5. Спроектировать и разработать многопользовательский веб-клиент для использования системы.
6. Провести сравнительный анализ качества ответов различных вариаций GraphRAG и RAG систем.
7. Оценить производительность и устойчивость системы, предложить направления дальнейшего развития.

Методологические основы проектирования RAG-систем и их реализации на языке программирования Python описаны в работах Д. Ротмана. В работах Т. Братанича и З. В. Апановича особое внимание уделяется использованию графа знаний для обработки текстовой информации. Разработка графовой базы данных Neo4j осуществлялась с опорой на пособие А. В. Маркина. Реализация веб-приложения велась согласно официальной документации используемых инструментов: фреймворка FastAPI, библиотек React, Sentence Transformers и Neo4j Graph Data Science Library.

Структура и объём работы. Бакалаврская работа состоит из введения, двух разделов, заключения, списка использованных источников и девяти приложений. Общий объем работы составляет 60 страниц, из них 44 страницы — основное содержание, включая 13 рисунков и 4 таблицы, цифровой носитель в качестве приложения и список использованных источников информации из 20 наименований.

ОСНОВНОЕ СОДЕРЖАНИЕ РАБОТЫ

Первый раздел «Анализ подходов к интеграции внешних знаний в большие языковые модели» посвящен исследованию ограничений LLM, связанных с особенностями их архитектуры, а также описанию принципов работы методов RAG и GraphRAG и присущих им недостатков.

В разделе рассматриваются архитектурные особенности больших языковых моделей, основанных на трансформере. Подробно описываются устройство и процесс обучения трансформера, в том числе механизмы многоголового внимания и самовнимания, позволяющие LLM учитывать контекст. Также анализируется тезис о том, что LLM не хранят знания в весах модели в явном виде, а функционируют как вероятностные модели, предсказывающие следующий токен в последовательности. Таким образом, из архитектуры трансформера выводятся присущие LLM ограничения, среди которых особенно выделяются «галлюцинации», устаревание информации и сложности дообучения готовых моделей.

В разделе приводятся два основных метода преодоления обозначенных ограничений больших языковых моделей: SFT (supervised fine-tuning) и RAG (retrieval-augmented generation). Ввиду большей актуальности и эффективности RAG-метода далее подробно рассматриваются принципы построения именно этого подхода.

Далее проводится обзор подхода генерации с дополненной выборкой (RAG), предполагающего использование внешних источников данных для повышения точности ответов LLM. В основе подхода лежит идея, согласно которой LLM выступает в роли инструмента обработки естественного языка, тогда как необходимая для ответа информация извлекается из внешней базы знаний.

Описываются основные этапы работы RAG-систем: индексирование документов, извлечение релевантных фрагментов и генерация ответа на их основе. В качестве внешней базы знаний используется векторное хранилище, в котором текстовые документы представлены в виде эмбеддингов текстовых чанков, то есть фрагментов фиксированного размера. Отмечается влияние длины чанков и степени их перекрытия на качество генерации ответов.

Приводится подробное описание работы поискового модуля RAG-системы, рассматриваются различия между семантическим и полнотекстовым поиском, а также их объединение в рамках гибридного подхода. В описании генератора

затрагивается вопрос выбора количества параметров используемой LLM.

Рассматриваются преимущества данного подхода, включая повышение фактической точности, наличие источников информации и возможность работы с актуальными данными, а также его ограничения, связанные с качеством поиска и сложностями при решении задач фильтрации и агрегации данных.

В качестве развития классического RAG анализируется подход GraphRAG, использующий графовые базы данных в качестве внешнего источника знаний. Описываются принципы построения графов знаний, их структура и способы применения для организации и извлечения информации. Рассматриваются два способа выделения триплетов из текста: с использованием LLM и с помощью эмпирических алгоритмов, основанных на синтаксической роли слов в предложении.

Особое внимание уделяется преимуществам графового представления данных, включая повышение связности контекста, улучшение навигации по информации и возможность более точного формирования ответов. Также рассматриваются недостатки данного подхода, такие как сложность построения и поддержки графа знаний, а также дополнительные вычислительные затраты.

В завершение раздела формулируются выводы о том, что использование внешних источников информации является необходимым условием повышения качества работы больших языковых моделей. На основании сравнительного анализа делается вывод о перспективности применения подхода GraphRAG как более структурированного и интерпретируемого по сравнению с классическим RAG, что обосновывает его выбор для дальнейшей разработки в рамках выпускной квалификационной работы.

Второй раздел «Разработка GraphRAG-приложения» посвящен программной реализации GraphRAG-системы и веб-приложения для доступа к ней, а также разработке классической RAG-системы и сравнительной оценки качества данных систем.

В ходе работы была реализована модель GraphRAG на языке программирования Python с подключением к графовой базе данных Neo4j и использованием LLM.

Система принимает на вход текст, извлеченный из корпуса документов. Текст предварительно разбивается на чанки по предложениям. Каждый чанк имеет фиксированный размер с частичным перекрытием. Для разбиения текста

на предложения используется библиотека spaCy с моделью `ru_core_news_lg`.

Для построения графа знаний на основе данного текста необходимо выделение из него триплетов, что реализуется классом `Triplets`. Для каждого чанка формируется промпт, который передается в LLM и содержит инструкции по извлечению фактов. Результат возвращается в формате JSON, что упрощает его дальнейшую обработку.

В приложении используется LLM `Mistral-Nemo-Instruct-2407`, содержащая 12 миллиардов параметров и поддерживающая большой контекст (до 128 тысяч токенов), что особенно важно при обработке длинных документов. Вызов модели осуществляется через HTTP API.

Дополнительно реализована процедура объединения различных наименований одной и той же сущности с целью повышения связности графа знаний. Сущности считаются эквивалентными, если скалярное произведение их эмбеддингов превышает заданный порог. В качестве эмбеддинг-модели используется `all-MiniLM-L6-v2` из библиотеки `SentenceTransformer`.

В качестве графовой базы данных выбрана СУБД Neo4j, предоставляющая открытый API, поддержку масштабируемых запросов и средства визуализации графов. Для взаимодействия с базой используется язык запросов Cypher.

В программной реализации класс `Neo4jConnection` использует библиотеку `neo4j` для подключения к базе данных, а класс `KnowledgeGraph` предоставляет методы для преобразования триплетов в графовое представление. При этом субъект и объект представляются узлами типа `Entity`, а предикат — направленной связью между ними, что реализовано в функции `insert_triplet`.

Каждый узел и отношение характеризуются двумя атрибутами: именем и эмбеддингом, причем для отношения используется эмбеддинг всего триплета.

Для извлечения релевантного контекста из графа применяется расширение Neo4j — библиотека Graph Data Science (GDS). В рамках работы данная библиотека используется для семантического поиска релевантных узлов и триплетов на основе эмбеддингов. GDS вычисляет косинусное расстояние между эмбеддингами и вектором пользовательского запроса, что позволяет определить наиболее близкие по смыслу сущности.

В текущей реализации по пользовательскому запросу класс `Search` извлекает из графа фиксированное количество узлов, после чего формируются все связанные с ними триплеты. Полученные триплеты сортируются по релевант-

ности, и top-k из них передаются генератору в качестве контекста. Извлечение связанных триплетов реализовано в функции `Search.get_related_triplets`.

Класс `Model` содержит метод `get_answer`, принимающий пользовательский запрос. Он осуществляет вызов процедуры извлечения релевантных триплетов из графа знаний с помощью функции `Search.get_related_triplets`, после чего сформированный контекст передается LLM вместе с соответствующей инструкцией для генерации ответа. Для проведения сравнительных экспериментов на языке программирования Python была реализована базовая версия классической RAG-модели.

Архитектура RAG-модели включает два основных класса (`RAGModel` и `RAG`) и состоит из нескольких этапов: подготовки и индексирования текстовых данных, поиска релевантного контекста и генерации ответа.

На этапе подготовки исходный текст обрабатывается методами `_chunk_text` и `_prepare` класса `RAG`. Текст разбивается на чанки по предложениям, аналогично алгоритму, использованному в GraphRAG. Каждый чанк преобразуется в векторное представление с использованием эмбеддинг-модели.

В качестве векторного хранилища используется библиотека `FAISS`, предназначенная для эффективного поиска ближайших соседей в векторном пространстве по косинусному сходству. Полученные векторы нормализуются и добавляются в индекс `FAISS`.

Метод `RAG.search` отвечает за поиск релевантного контекста. При обработке пользовательского запроса он также преобразуется в вектор, после чего выполняется поиск наиболее близких по смыслу чанков в индексе. В реализации используется выборка top-k наиболее релевантных фрагментов.

Метод `RAGModel.get_answer` объединяет описанные этапы: извлекает релевантный контекст и передает его в языковую модель. Для генерации ответа формируется специальный промпт, содержащий инструкции, ограничивающие поведение модели и повышающие качество получаемых результатов.

Для взаимодействия с RAG-системами был разработан веб-клиент, выступающий промежуточным звеном между пользовательскими запросами и серверной частью. Приложение поддерживает многопользовательский режим, позволяет сохранять историю запросов и обеспечивает независимую работу с различными наборами документов.

К основным функциональным возможностям веб-приложения относятся

регистрация и аутентификация пользователей, создание и управление чатами, отправка текстовых запросов, загрузка и обработка пользовательских документов, хранение истории сообщений, а также получение ответов, сгенерированных на основе загруженных данных с использованием подхода GraphRAG.

Веб-приложение реализовано в рамках клиент-серверной архитектуры: клиентская часть разработана с использованием фреймворка React (JavaScript), серверная — фреймворка FastAPI (Python). Взаимодействие между компонентами осуществляется по протоколу HTTP с использованием REST API.

Разработанный API можно разделить на две группы: аутентификация пользователей и работа с чатами и документами. К первой группе относятся методы `/register` и `/login`, обеспечивающие регистрацию и аутентификацию пользователя. Результатом их работы является, в том числе, генерация JWT-токена, используемого для авторизации при обращении к остальным методам API.

Взаимодействие компонентов системы инициируется клиентской частью, которая отправляет HTTP-запросы на сервер FastAPI. При авторизации выполняется запрос `POST /login`, в ответ на который пользователь получает токен доступа, сохраняемый в системе. При отправке сообщения формируется запрос `POST /chat`, который обрабатывается сервером с возможным обращением к модулю GraphRAG для генерации ответа. Для загрузки файлов используется запрос `POST /upload`, в ходе которого документ обрабатывается, его содержимое добавляется в пользовательские данные, а история чата обновляется.

Серверная часть реализует указанные API-эндпоинты и взаимодействует с реляционной базой данных PostgreSQL, используемой для хранения информации о пользователях, чатах и истории сообщений. Работа с базой данных осуществляется через ORM-библиотеку SQLAlchemy, что позволяет описывать структуры данных без явного использования SQL-запросов.

В рамках модели данных определены две сущности: пользователь (User) и чат (Chat), связанные отношением «один-ко-многим». Таблица `users` содержит уникальный идентификатор пользователя и хеш пароля, а таблица `chats` включает идентификатор чата, его название, внешний ключ пользователя и поле `messages`, представляющее историю сообщений в формате JSON. Использование JSON обеспечивает гибкость хранения и быстрый доступ к данным, а индексация ключевых полей ускоряет поиск.

API-эндпоинты реализованы средствами FastAPI с использованием объекта `APIRouter` для логической группировки маршрутов. Методы `/chats` и `/history` позволяют получать список чатов и историю сообщений, `/create_chat` создает новые чаты, а `/upload` — загружать и обрабатывать документы. Основной метод `/chat` реализует логику обработки пользовательского запроса: выполняет проверку наличия чата, сохраняет сообщения и, при наличии документов, инициирует построение или обновление графа знаний. Ответ формируется с использованием модуля GraphRAG.

Для обеспечения многопользовательского режима GraphRAG расширен добавлением идентификатора чата к узлам графа, что позволяет изолировать данные разных пользователей.

Клиентская часть реализована как одностраничное приложение (SPA) на базе React, что обеспечивает динамическое обновление интерфейса без перезагрузки страницы и снижает нагрузку на сервер. Архитектура построена на компонентном подходе.

Основными компонентами интерфейса являются `AuthModal` (аутентификация), `ChatSidebar` (список чатов), `ChatContainer` (диалог) и `App`, координирующий работу приложения. Компоненты обеспечивают ввод и отображение данных, взаимодействие с пользователем и отправку запросов к серверу.

Для управления состоянием используются пользовательские хуки `useAuth` и `useChats`. Первый отвечает за аутентификацию, хранение пользователя и JWT-токена, обеспечивая сохранение сессии, второй — за работу с чатами, включая загрузку списка, получение истории и создание новых чатов. Взаимодействие с API вынесено в отдельный сервисный слой, выполняющий HTTP-запросы и обработку токена.

Оценка качества и надежности RAG-моделей. Для оценки разработанных моделей RAG и GraphRAG был проведен эксперимент на пользовательском наборе данных.

В качестве датасета использовались тексты из статей русскоязычной «Википедии» объёмом более 50 000 символов по темам LLM и RAG.

На основе данных текстов был сформирован набор из 50 вопросов различной сложности: простые вопросы, требующие извлечения одного факта, и сложные вопросы, требующие объединения нескольких фактов и установления связей между ними. Для каждого вопроса был заранее подготовлен эталонный

ответ (англ. ground truth).

В эксперименте участвовали 5 моделей:

1. GraphRAG с поиском релевантных узлов в графе знаний и извлечением связанных с ними триплетов (GraphRAG-nodes);
2. GraphRAG с поиском релевантных триплетов в графе знаний (GraphRAG-triplets);
3. RAG с поиском 3-х релевантных чанков (RAG-3);
4. RAG с поиском 6-и релевантных чанков (RAG-6);
5. LLM без подключения к внешней базе знаний.

Каждая из моделей ответила на 50 вопросов. Полученные ответы оценивались с использованием LLM на основе вопроса, извлеченного контекста и эталонного ответа по шкале от 0 до 10 по трём метрикам: релевантность контекста (`context_relevance`), достоверность (`context_usage`) и точность ответа (`answer_accuracy`).

Численные результаты эксперимента представлены на рисунках 1 и 2.

Из графиков видно, что GraphRAG-модели извлекают более релевантный контекст. RAG-модели имеют более низкие значения метрик `context_relevance` и `context_usage`, что обусловлено наличием шума в извлекаемых чанках: из-за их размера они могут содержать частично нерелевантную информацию.

Модель RAG-6 незначительно уступает RAG-3 по качеству извлеченного контекста, что приводит к снижению точности ответов. Данный эффект особенно заметен на простых вопросах, для которых увеличение числа чанков повышает вероятность включения частично релевантных фрагментов.

Модель GraphRAG-triplets демонстрирует наилучшую среднюю точность ответов и извлекает более релевантный контент по сравнению с GraphRAG-nodes. Она значительно лучше справляется с простыми вопросами, что можно объяснить тем, что при узловом поиске избыточное количество связанных узлов может приводить к включению лишнего контекста.

Хотя GraphRAG-triplets уступает RAG-3 в точности ответов на простые вопросы, GraphRAG-модели демонстрируют лучшие результаты на сложных вопросах — разница метрик превышает 1 пункт. Для сложных вопросов RAG-модели показывают результаты, сопоставимые с обычной LLM, не использующей внешний контекст.

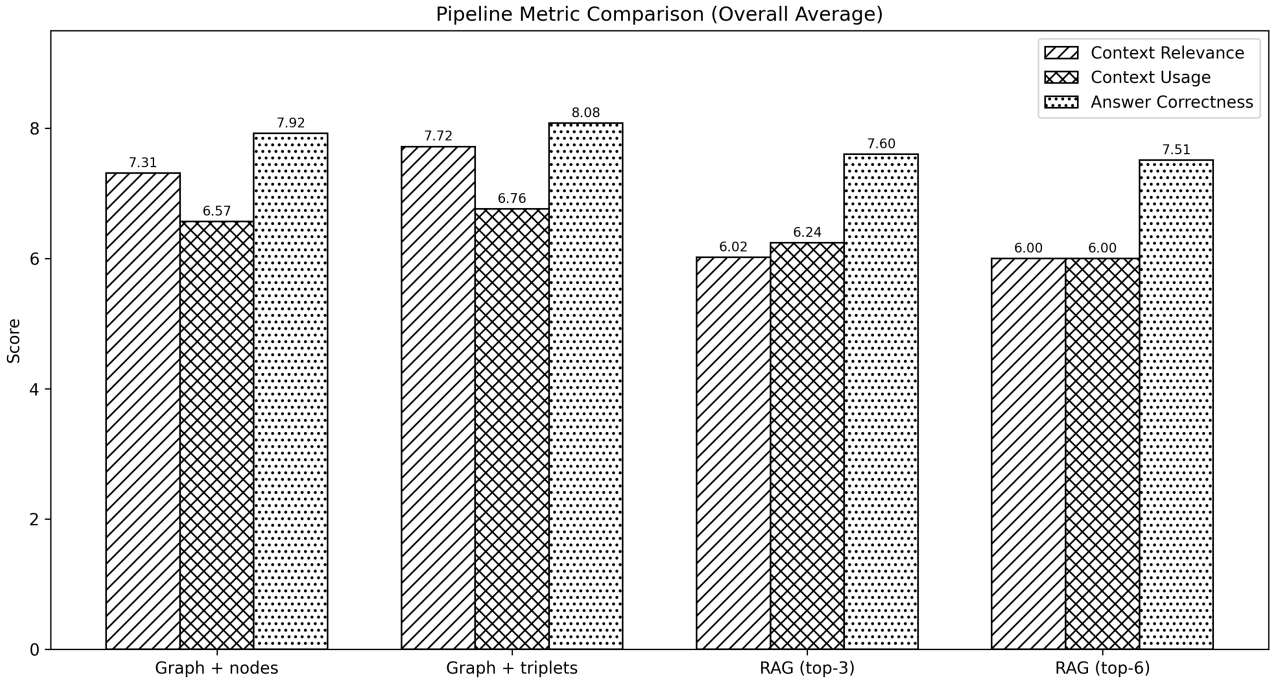

Рисунок 1 – График сравнения метрик RAG-моделей

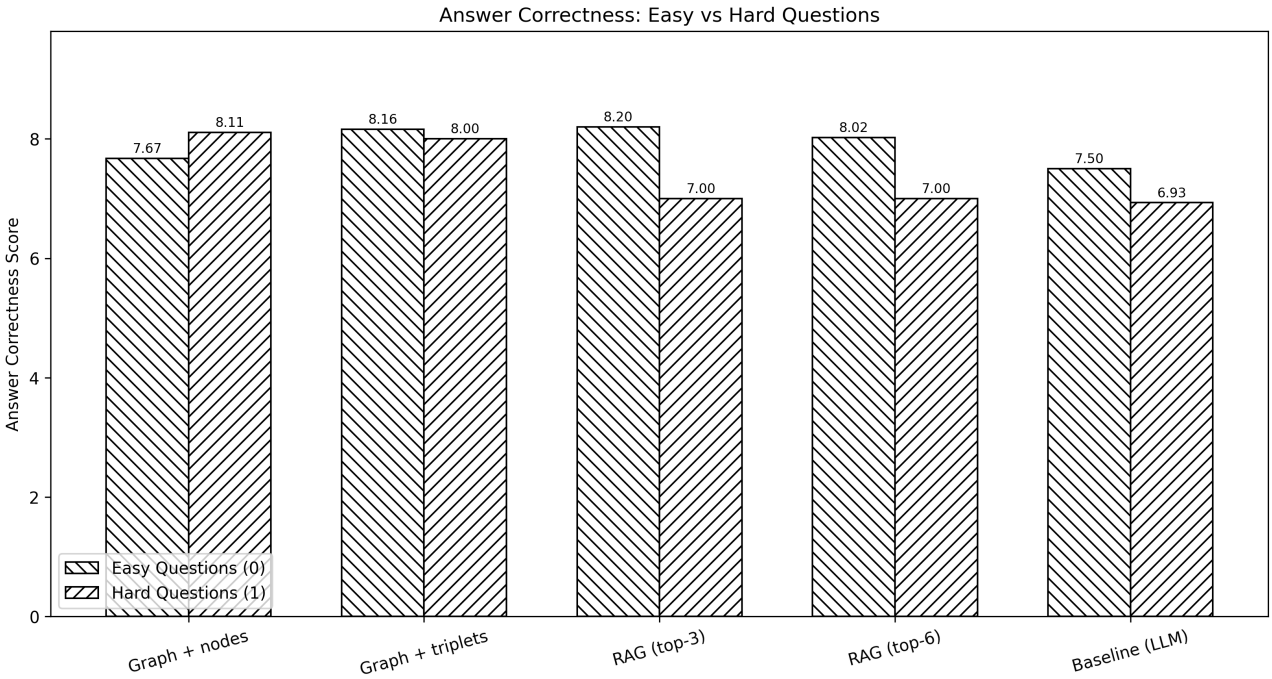

Рисунок 2 – График сравнения точности ответов RAG-моделей и LLM в разрезе простых и сложных вопросов

Для RAG-моделей основной причиной неполных ответов является отсутствие релевантного контекста в результате поиска. В частности, при ответах на вопросы о причинах явлений поисковый модуль может извлекать информацию об их следствиях. Для повышения качества возможно применение более продвинутых методов ранжирования и фильтрации извлеченного контента.

ЗАКЛЮЧЕНИЕ

В рамках данной работы была спроектирована, программно реализована и экспериментально исследована система генерации ответов на основе пользовательской документации с применением метода GraphRAG, ориентированная на обработку русскоязычных текстов. В ходе исследования все поставленные задачи были решены в полном объеме, что позволило достичь заявленной цели работы.

К основным научным и практическим результатам работы относятся следующие положения.

1. Проведен теоретический анализ архитектуры трансформера и принципов работы больших языковых моделей. Также описаны их ключевые ограничения, такие как склонность к «галлюцинациям» и высокая стоимость актуализации внутренних знаний.
2. Разработана классическая RAG-система на основе векторного индекса библиотеки FAISS, а также усовершенствованная модель GraphRAG, интегрированная с графовой базой данных Neo4j.
3. Реализовано клиент-серверное веб-приложение в многопользовательском режиме на базе современных фреймворков FastAPI и React.
4. Выполнена комплексная оценка качества генерации ответов по методологии LLM-as-a-Judge на сформированном русскоязычном датасете. Экспериментально подтверждено, что метод GraphRAG обладает преимуществом при обработке сложных (multi-hop) запросов, требующих установления структурных связей между сущностями из разных фрагментов текста. Классический RAG демонстрирует меньшую временную задержку и высокую точность на простых вопросах о фактах.

На основе выявленных ограничений реализованных моделей были сформулированы направления дальнейшего развития разработанной системы. Для оптимизации ресурсов и достижения наибольшей точности ответа возможна предварительная классификация сложности пользовательского запроса. На основе такой классификации будет осуществляться дальнейшей выбор RAG-модели. Развитие данной идеи предполагает внедрение ИИ-агентов в систему.