СОДЕРЖАНИЕ

[ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ 3](#_Toc197916680)

[ВВЕДЕНИЕ 4](#_Toc197916681)

[1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ RETRIEVAL AUGMENTED GENERATION 6](#_Toc197916682)

[1.1 Ключевые этапы RAG пайплайна 7](#_Toc197916683)

[1.1.1 Chunking 7](#_Toc197916684)

[1.1.2 Retrieval 8](#_Toc197916685)

[2 ПРОДВИНУТЫЕ ПОДХОДЫ 9](#_Toc197916686)

[2.1 Chunking 9](#_Toc197916687)

[2.1.1 Семантический Chunking 9](#_Toc197916688)

[2.2 Query Rewriting 11](#_Toc197916689)

[2.2.1 HyDE 11](#_Toc197916690)

[2.3 Reranking 12](#_Toc197916691)

[2.4 Agentic RAG 13](#_Toc197916692)

[3 ТОЧКИ ОТКАЗА СИСТЕМ С RETRIEVAL AUGMENTED GENERATION 15](#_Toc197916693)

[4 ОЦЕНКА СИСТЕМ С RETRIEVAL AUGMENTED GENERATION 17](#_Toc197916694)

[4.1 QASPER 17](#_Toc197916695)

[4.2 RAGAS 18](#_Toc197916696)

[4.3 Основные метрики 18](#_Toc197916697)

[4.3.1 Faithfulness 18](#_Toc197916698)

[4.3.2 Response Relevancy 19](#_Toc197916699)

[4.3.3 Answer Correctness 19](#_Toc197916700)

[5 ПРОЕКТИРОВАНИЕ СИСТЕМЫ 21](#_Toc197916701)

[5.1 Дизайн-документ 21](#_Toc197916702)

[5.2 Постановка задачи 21](#_Toc197916703)

[5.3 Архитектура серверной части 22](#_Toc197916704)

[5.3.1 Хранение данных 23](#_Toc197916705)

[5.4 Клиент 24](#_Toc197916706)

[5.5 Пилотный запуск 24](#_Toc197916707)

[5.6 Требования к работе системы 25](#_Toc197916708)

[5.6.1 Механизмы безопасности 26](#_Toc197916709)

[5.7 UML-диаграммы 26](#_Toc197916710)

[5.7.1 Диаграмма вариантов использования (Use Case) 26](#_Toc197916711)

[5.7.2 Диаграмма последовательностей (Main Sequence) 26](#_Toc197916712)

[5.7.3 Дополнение: Диаграмма активностей (User Activity) 27](#_Toc197916713)

[6 РЕАЛИЗАЦИЯ ПРОТОТИПА 29](#_Toc197916714)

[6.1 Общая структура системы 30](#_Toc197916715)

[6.2 Серверная часть 30](#_Toc197916716)

[6.3 Итоговая архитектура 30](#_Toc197916717)

[6.4 Клиентская часть 33](#_Toc197916718)

[6.5 Хранение данных 34](#_Toc197916719)

[6.6 Модуль Retrieval-Augmented-Generation 34](#_Toc197916720)

[7 ТЕСТИРОВАНИЕ 36](#_Toc197916721)

[7.1 Датасет 36](#_Toc197916722)

[7.2 Конфигурации 37](#_Toc197916723)

[7.3 Анализ результатов 37](#_Toc197916724)

[7.3.1 Сравнение метрик на разных конфигурациях 38](#_Toc197916725)

[7.3.2 Выводы по метрикам 38](#_Toc197916726)

[7.4 Пилотное тестирование 38](#_Toc197916727)

[ЗАКЛЮЧЕНИЕ 40](#_Toc197916728)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 42](#_Toc197916729)

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

LLM (Large Language Model, Большая языковая модель) – большая языковая модель, обученная на текстовых данных, используемая для генерации ответов на запросы пользователя.

Бенчмарк – набор данных для вычисления метрик работоспособности системы в различных сценариях.

Эмбеддинг-модель – модель, преобразующая текстовые данные в векторное представление.

Промпт – запрос в большую языковую модель.

Пайплайн

ВВЕДЕНИЕ

В современном мире темп технологического прогресса и количество научных публикаций растут с каждым днём, и эффективное освоение научной литературы становится фактором успешной работы в научной и инженерной сферах. Особенно это актуально для специалистов в области информационных технологий. Доступ к научным публикациям зачастую ограничивается не только объёмом информации, но и её формой представления: большинство современных научных статей публикуется на английском языке и содержат много терминов и абстракций.

Задачи, связанные с систематическим изучением больших массивов научной информации, требуют от исследователей значительных временных затрат на поиск, фильтрацию, чтение и осмысление данных. В условиях информационной перегрузки традиционные методы работы с текстами (ручной перевод, чтение и поиск литературы) становятся всё менее эффективными. Это делает актуальным использование искусственного интеллекта для упрощения ключевых этапов взаимодействия человека с научной информацией.

Одним из перспективных направлений является подход Retrieval-Augmented-Generation (RAG), совмещающий механизмы информационного поиска с возможностями больших языковых моделей. RAG позволяет не только находить необходимую информацию в больших корпусах документов, но и формировать на её основе осмысленные и понятные ответы, включая пояснения терминов и перевод.

Целью выпускной квалификационной работы является разработка системы, использующей подход Retrieval-Augmented-Generation для помощи пользователям в изучении учебных материалов, в частности научных статей формате PDF. Система должна обеспечивать автоматический поиск и извлечение релевантной информации из предоставленного документа, пояснение встречающихся терминов, перевод с английского языка, а также ответы на вопросы пользователя по документу. Предполагается, что подобный инструмент позволит повысить эффективность изучения научной литературы студентами, снизить языковые и когнитивные барьеры, а также сократить время на обработку и усвоение информации.

Для реализации поставленной цели были сформулированы следующие задачи:

1. Изучить современные подходы к построению систем на основе Retrieval-Augmented-Generation, определить их применимость к поставленной задаче.
2. Разработать интуитивно понятный и удобный пользовательский интерфейс для взаимодействия с системой.
3. Спроектировать и реализовать сервис на основе Retrieval-Augmented-Generation и обеспечить взаимодействие с пользователем в реальном времени.
4. Реализовать алгоритм оценки качества системы с точки зрения полноты и точности предоставляемых ответов.
5. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ RETRIEVAL AUGMENTED GENERATION

С развитием технологий обработки естественного языка большие языковые модели продемонстрировали выдающиеся результаты в различных задачах, включая генерацию текста, машинный перевод, вопросно-ответные системы и обобщение информации. Большие языковые модели обладают способностью усваивать огромные объёмы информации из обучающих данных, функционируя как неявная база знаний. Однако такая архитектура имеет существенные ограничения.

Во-первых, языковые модели обучаются на статических наборах данных и не могут обновлять свои знания без полного переобучения. Это делает их уязвимыми к устареванию информации. Во-вторых, в отсутствии внешнего источника верификации, модели склонны к галлюцинациям (генерации некорректной или вымышленной информации). Эти проблемы особенно критичны в задачах, где требуется точность, достоверность и актуальность ответов, например, при работе с научными текстами.

Для преодоления этих ограничений был предложен подход Retrieval-Augmented-Generation (RAG), сочетающий в себе мощь больших языковых моделей и возможности внешнего поиска по базе знаний. Основная идея RAG заключается в том, чтобы обогатить запрос пользователя дополнительным контекстом, и только затем передать его на вход генеративной модели. Таким образом, RAG позволяет обеспечить более достоверные, обоснованные и контекстуально релевантные ответы.

На **Рисунке 1** представлена обобщённая архитектура RAG-пайплайна. Он включает два ключевых этапа: **поиск релевантной информации (Retrieve)** и **генерацию ответа на её основе (Generate)**. На первом этапе система извлекает из внешнего хранилища фрагменты, наиболее подходящие под запрос. Затем эти фрагменты вместе с оригинальным запросом подаются на вход большой языковой модели, которая генерирует ответ, используя как внутренние знания, так и предоставленный контекст.

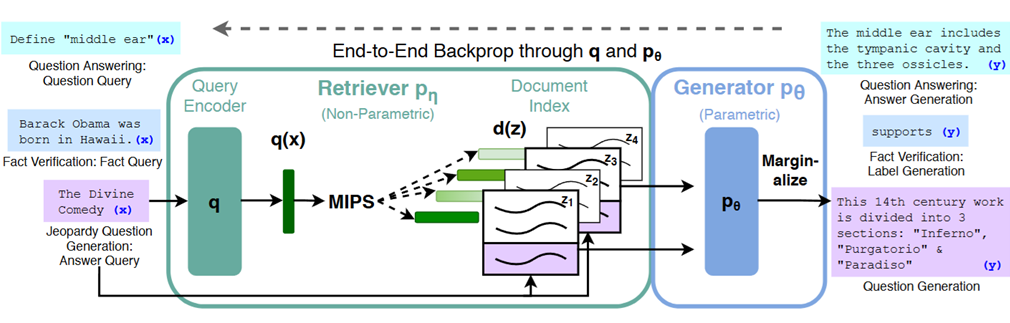
****

Рисунок 1 - Устройство пайплайнаRetrieval Augmented Generation

* 1. Ключевые этапы RAG пайплайна

Подход RAG состоит из ряда модульных этапов, каждый из которых играет важную роль в обеспечении качества итогового ответа. Рассмотрим их подробнее.

* + 1. Chunking

Этап **Chunking** (разбиение текста на фрагменты) представляет собой начальный этап подготовки корпуса для поиска. Исходные документы разбиваются на небольшие логически связанные **чанки**. Основная задача получить фрагменты, которые одновременно:

1. содержат достаточное количество информации для понимания контекста
2. не являются слишком длинными, чтобы их можно было эффективно обрабатывать языковой моделью
3. сохраняют логическую и смысловую связность

Существует два основных подхода к чанкингу:

1. **Эвристический подход**, основанный на структурных признаках текста. В качестве границ чанков используются элементы форматирования: абзацы, заголовки, списки, таблицы, а также знаки препинания. Такой подход прост в реализации и даёт хорошие результаты при наличии чётко структурированных документов.
2. Семантический подход, ориентированный на сохранение смысловой целостности фрагментов. Здесь применяются методы на основе векторных представлений текста: текст сначала представляется в виде последовательности векторов, а затем разбивается так, чтобы внутри чанков максимизировалось семантическое сходство, а между ними минимизировалось.

Выбор подхода зависит от специфики задач и характеристик документов: для технических текстов зачастую достаточно эвристик, в то время как для художественных или слабоформализованных текстов предпочтительнее использовать семантический подход.

* + 1. Retrieval

На этапе Retrieval происходит поиск релевантных к запросу фрагментов текста. Существуют два основных подхода к построению Retrieval-этапа:

1. Традиционные методы информационного поиска, такие как триграммный поиск, TF-IDF и BM25 [ссылка]. Эти методы быстрые, но они не учитывают смысл текста.
2. Методы на основе эмбеддингов. Здесь как запрос, так и текстовые фрагменты кодируются в векторном пространстве с помощью эмбеддинг модели. Сходство между векторами измеряется с помощью косинусного расстояния или других метрик. Такой подход учитывает смысл текста, а также он устойчив к опечаткам и синонимам.

Для Retrieval этапа зачастую применяются гибридные схемы, объединяющие преимущества обоих подходов. Например, можно сначала отобрать 100 кандидатов с помощью BM25, а затем провести их переранжирование с использованием эмбеддинг моделей.

1. ПРОДВИНУТЫЕ ПОДХОДЫ

С момента появления Retrieval-Augmented-Generation было предложено множество подходов для повышения его эффективности. Эти улучшения затрагивают практически все стадии пайплайна. Далее я рассмотрю некоторые усовершенствования, сгруппированные по соответствующим этапам.

* 1. Chunking

Чанкинг является ключевым этапом подготовки корпуса текста. Главная задача на этом этапе добиться оптимального баланса между объемом фрагмента и сохранением его смысловой целостности. Это особенно важно в доменно-специфичных системах [ссылка], где плотность информации может значительно варьироваться. В идеале чанк должен представлять собой логически завершённую мысль, которая не теряет смысла вне контекста. Нарушение границ предложений или смысловых блоков может привести к потере критически важной информации. Оптимально, если чанк охватывает не отдельные предложения, а их логически связанные группы.

* + 1. Семантический Chunking

1. **Данный метод [ссылка] использует эмбеддинг-модели для группировки связанных по смыслу предложений. В зависимости от имплементаций алгоритм может меняться, но в общем виде он реализуется в два этапа. Первым этапом делят текст на предложения и вычисляют их векторные представления. Затем между соседними парами предложений вычисляют косинусные расстояния, в результате чего получается распределение значений (пример на Рисунке 2). Наконец, текст разделяется по пороговому значению, например, выбирается 95-й процентиль по получившемуся распределению (красная линия на Рисунке 2).**

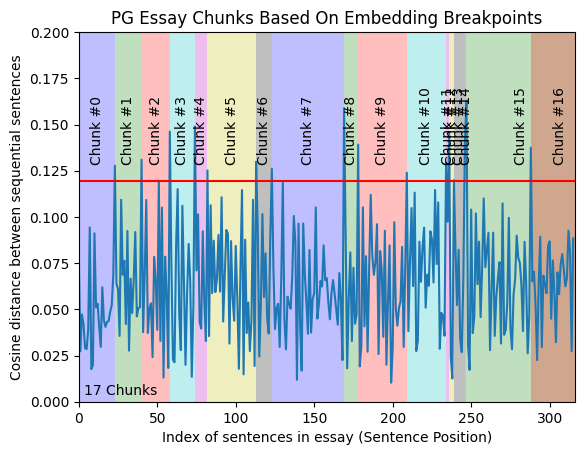
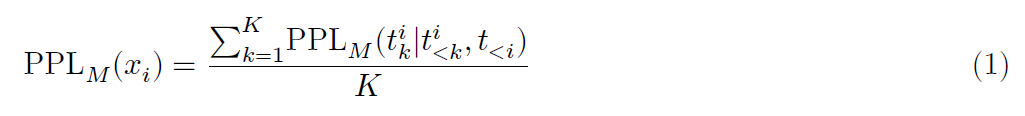


Рисунок 2 – распределение косинусных расстояний

1. Perplexity Chunking.Цель данного метода - сформировать набор чанков (X1, X2, ..., Xk), где каждый чанк представляет собой логически связанное объединение исходных предложений. На начальном этапе текст разделяется на набор предложений (x1, x2, ..., xn). Для объединения исходных предложений в чанки модель вычисляет перплексию (PPL) для каждого предложения на основе предшествующих предложений:



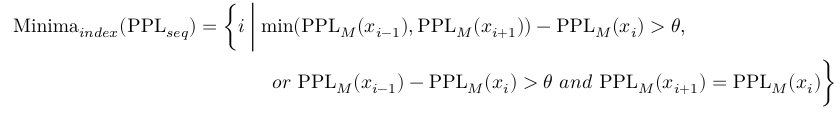
(1)

где K - общее количество токенов в ​, ​ - k-й токен в ​, а обозначает все токены, предшествующие ​. Для определения границ чанков алгоритм анализирует последовательность:



(2)

В поисках минимальных значений:



(3)

Смысл данной формулы заключается в следующем: эти минимумы рассматриваются как потенциальные границы фрагментов в одном из двух случаев:

1. Значения PPL по обе стороны точки выше, чем в самой точке, и разница хотя бы с одной стороны превышает заданный порог θ.
2. Разница между левой точкой и текущей больше θ, а значение справа равно значению в текущей точке.
   1. Query Rewriting

Цель этапа Rewriting заключается в адаптации пользовательского запроса к структуре и стилю данных, хранящихся в индексе. Это особенно важно, если пользователь использует в своем запросе общие или неоднозначные формулировки, которые не соответствуют документам в целевом корпусе.

* + 1. HyDE

Метод HyDE предполагает генерацию гипотетического документа на основе запроса. Языковая модель получает похожую инструкцию: “Напиши документ, который мог бы быть ответом на данный вопрос”. Сгенерированный документ не обязательно достоверен, он может содержать фактические ошибки, так как его задача лишь имитировать релевантный текст. Ожидается, что в результате кодирования документа с помощью эмбеддинг модели удалятся лишние или недостоверные детали и в результате полученный вектор станет семантически ближе к релевантным фрагментам, чем вектор оригинального запроса пользователя.

* 1. Reranking

После первоначального извлечения релевантных документов на этапе Retrieval (например, с помощью алгоритма BM25 [ссылка]), часто применяются **алгоритмы переранжирования (этап Reranking)**, уточняют порядок релевантности результатов. Рассмотрим два наиболее распространенных подхода:

1. Bi-Encoder. Запрос и каждый документ обрабатываются раздельно с помощью эмбеддинг моделей (энкодеры на Рисунке 2). На выходе получаются векторы, между которыми вычисляется косинусное сходство. На Рисунке 2 показано, как тексты преобразуются в векторы, объединяются в итоговые вектора (Pooling), а в конце вычисляется их сходство.

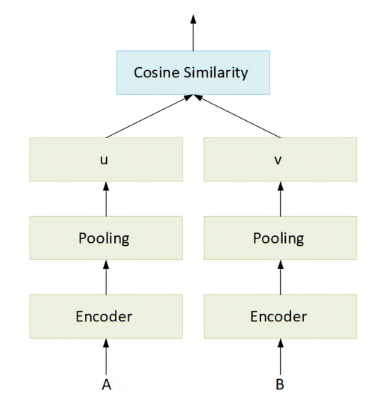


Рисунок 2 — Bi-Encoder

1. Cross-Encoder. Запрос и документ объединяются в одну строку и подаются на вход языковой модели. Модель рассматривает их взаимодействие и возвращает итоговую оценку релевантности (similarity score) напрямую. Этот метод более ресурсоёмкий, но значительно точнее. На Рисунке 3 изображена последовательность преобразований. Similarity Score – выходной слой (например, полносвязный слой с сигмоидной функцией активации), который выдает оценку сходства: число в диапазоне от 0 до 1.

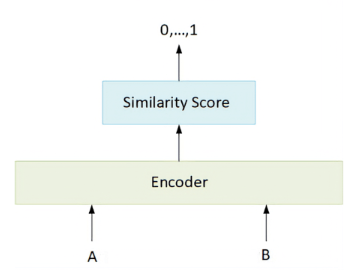


Рисунок 3 — Cross-Encoder

* 1. Agentic RAG

Agentic RAG представляет собой усовершенствованный подход к построению Retrieval-Augmented-Generation. В отличие от традиционного RAG, где модель генерирует ответы на основе единственного шага извлечения данных, Agentic RAG позволяет большой языковой модели самостоятельно формулировать уточняющие запросы, критически оценивать полученные результаты и, при необходимости, повторно обращаться к источникам. Более того, такие системы предполагают использование произвольных инструментов – внешних функций, к которым имеет доступ большая языковая модель. Такие инструменты работают в пользовательском окружении для реализации разных задач, например веб-поиска, математических вычислений, доступа к базе данных и многого другого.

Один из наиболее известных подходов к построению агентных систем описан в статье ReAct [ссылка]. Он сочетает рассуждение (Reasoning) и действие (Acting). В отличие от простого вызова инструментов, ReAct заставляет модель вести цепочку размышлений, которая помогает ей выбирать нужные инструменты и приходить к финальному ответу. В ReAct происходит такая цепочка рассуждений и действий (см. Рисунок 3):

1. Thought (рассуждение): модель генерирует шаг рассуждения, что ей нужно сделать на шаге действия и почему.
2. Action (действие): на основе шага рассуждения модель сама вызывает инструмент, подставляя в него параметры, либо вызывает генерацию финального ответа.
3. Observation (наблюдение, вывод) Получив результаты инструмента, модель делает выводы, и цикл повторяется.

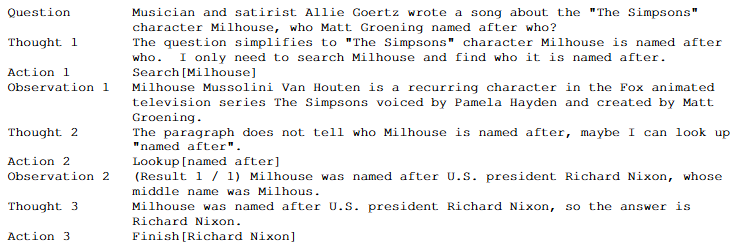


Рисунок 3 – пример цепочки рассуждений ReAct.

Для реализации агентных систем такого рода можно использовать фреймворк smolagent. Он предоставляет удобный способ задавать инструменты для модели, а внутри реализует алгоритм, описанный на рисунке 4, где все начинается с пользовательского запроса, а далее в цикле происходит цепочка размышлений-действий, подобная описанной в статье про ReAct [ссылка].

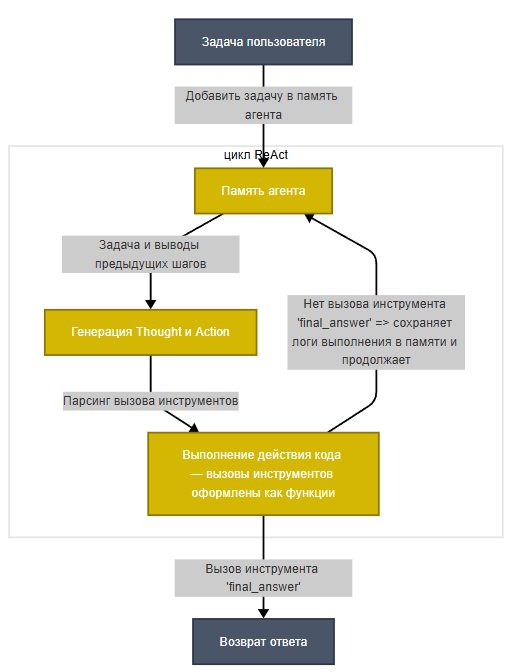


Рисунок 4 – Цикл выполнения smolagents.

1. ТОЧКИ ОТКАЗА СИСТЕМ С RETRIEVAL AUGMENTED GENERATION

Системы Retrieval-Augmented Generation (RAG) обладают значительным потенциалом в задачах генерации ответов с опорой на внешние источники знаний. Однако стоит отметить, что при проектировании и эксплуатации подобных систем возникает ряд характерных точек отказа, способных существенно повлиять на качество выдачи. В статье “Seven Failure Points When Engineering a Retrieval Augmented Generation System” [ссылка] авторы выделяют семь наиболее распространённых проблем, с которыми сталкиваются разработчики:

1. Отсутствие необходимого контента. Даже при корректной работе всех компонентов, системе может не удаваться извлечь релевантную информацию, если она отсутствует в базе знаний. Это может быть обусловлено неполнотой корпуса документов или ограничениями алгоритмов индексации.
2. Пропуск высокоранжированных результатов. Алгоритмы извлечения или ранжирования могут не отобрать релевантные фрагменты, даже если они присутствуют в индексе. Такое поведение часто связано с недостаточной точностью эмбеддинг моделей, неправильно подобранными мерами векторной близости или ошибками в алгоритмах поиска.
3. Неправильный контент. Извлечённые документы могут не соответствовать исходному запросу. Это приводит к генерации некорректных или вводящих в заблуждение ответов, особенно в случаях, когда модель генерации некритично воспринимает входные данные.
4. Неподходящий формат: Информация может быть представлена в формате, который затрудняет ее обработку или интеграцию в ответ, например, в виде изображений или таблиц.
5. Неправильная степень специфичности: Ответы, формируемые моделью, могут быть либо чрезмерно обобщёнными, либо излишне детализированными. Это снижает их полезность и не соответствует ожиданиям пользователя.
6. Неполные ответы: Система может предоставлять ответы, содержащие лишь часть необходимой информации, даже если полный ответ доступен в предоставленных документах.
7. Необходимость постоянной калибровки: Системы RAG подвержены деградации качества со временем и требуют периодической переоценки ключевых компонентов. Особенно это критично при изменении пользовательских предпочтений или обновлении источников данных.
8. ОЦЕНКА СИСТЕМ С RETRIEVAL AUGMENTED GENERATION

Как было показано в предыдущей главе, при проектировании и внедрении систем с Retrieval-Augmented-Generation возникает множество потенциальных точек отказа. Для своевременного выявления этих уязвимостей и повышения качества ответов системы необходимы надежные методы оценки. Поскольку RAG объединяет в себе несколько компонентов, методы оценки должны быть комплексными, охватывающими как отдельные этапы пайплайна, так и работу всей системы в целом.

В обзорной статье “Evaluation of Retrieval-Augmented Generation: A Survey” авторы выделяют три ключевых подхода к оценке RAG-систем:

1. **Оценка человеком** зачастую наиболее точный метод, который позволяет экспертам выявить такие нюансы оценки, которые сложно автоматизировать: смысла, стиля и релевантности контекста. Однако такой метод требует сравнительно больших ресурсов и времени.
2. **Фреймворки и датасеты** предоставляют воспроизводимые, масштабируемые и автоматизированные способы оценки. Они гораздо дешевле и быстрее, чем оценка человеком, но требуют серьезный анализ при подборе метрик и датасетов. Без тщательной настройки такие инструменты могут давать искаженные или недостаточно информативные результаты.

Кроме того, в статье представлена сводная таблица фреймворков и инструментов для оценки RAG-систем. Помимо RAGAS и MultiHop-RAG, о которых пойдет речь дальше, можно выделить датасет QASPER, как решение, подходящее по области вопросов.

* 1. QASPER

**QASPER** — это специализированный датасет, разработанный для задач вопросно-ответного поиска по научным публикациям в области обработки естественного языка. Он включает в себя **5049** вопросов, сформулированных по **1585-и** NLP-статьям. Вопросы создавались двумя группами экспертов: первая группа составляла вопросы, опираясь только на заголовок и аннотацию статьи, а вторая отвечала на них, используя полный текст статьи.

Особенность данного датасета состоит в том, что для ответа на вопросы используется только один документ. Несмотря на ценность QASPER, для RAG-систем, способных работать с несколькими источниками, требуется расширение или использование других датасетов, ориентированных на **multi-hop** ответы.

* 1. RAGAS

RAGAS [6] — это фреймворк с открытым исходным кодом, разработанный специально для оценки качества работы RAG-систем. Он предоставляет как средства для **генерации тестового датасета**, так и **набор метрик** для поэтапной оценки компонентов системы. Метрики для отдельных частей RAG позволяют декомпозировать оценку пайплайна на оценку его составляющих. С помощью таких метрик проще отслеживать уязвимые места системы, а также оценивать реакцию на изменение отдельных компонентов.

Метрики RAGAS основаны на подходе **LLM-as-a-Judge**, то есть большая языковая модель используется в качестве судьи: ее задача заключается в оценке качества работы системы, имитируя человеческую оценку. Далее рассмотрим некоторые из этих метрик более подробно.

* 1. Основные метрики

Для детальной оценки качества ответов в RAGAS (и других системах) применяются следующие метрики:

* + 1. Faithfulness

Метрика достоверности (Faithfulness [3]) измеряет, насколько фактически точен ответ относительно извлеченного контекста. Значения варьируются от 0 до 1, где 1 означает, что утверждения в ответе подтверждаются контекстом.

Алгоритм расчета метрики:

1. Разбиение ответа на утверждения.
2. Каждое утверждение проходит проверку на релевантность к извлеченному контексту.
3. Вычисление показателя достоверности по формуле:

(1)

* + 1. Response Relevancy

Метрика релевантности ответа (Response Relevancy [3])оценивает то, насколько ответ соответствует пользовательскому запросу. Высокий балл означает, что ответ полноценно отвечает на запрос без избыточной или нерелевантной информации.

Алгоритм расчета метрики:

1. Генерация 3-х искусственных вопросов на основе ответа системы.
2. Вычисление косинусного сходства между векторными представлениями запроса и каждого из вопросов.
3. Усреднение полученных значений.
   * 1. Answer Correctness

Метрика корректности ответа (Answer Correctness [3]) оценивает соответствие ответа эталонному. Балл варьируется от 0 до 1, где 1 означает полное совпадение.

Алгоритм расчета метрики:

1. Оценка фактической корректности: измеряет степень совпадения фактов между сгенерированным ответом и эталонным. Рассчитывается, как f1-score:

(2)

где TP – количество утверждений, присутствующих и в эталонном, и в сгенерированном ответах. FP – количество утверждений, которые есть в сгенерированном ответе, но отсутствуют в эталонном. FN – количество утверждений, которые есть в эталонном ответе, но отсутствуют в сгенерированном.

1. Оценка семантического сходства: рассчитывается на основе косинусной близости эмбеддингов сгенерированного и эталонного ответов.
2. Ответ: рассчитывается, как взвешенное среднее между фактическим соответствием и семантическим сходством
3. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Перед написанием кода проекта я решил углубиться в проектирование архитектуры системы. Целью данного этапа было уточнение требований, функционала и инструментов, используемых при разработке.

Для проектирования я активно использовал средства визуального моделирования, в частности UML-диаграммы. Они позволили формализовать и структурировать требования к системе на раннем этапе, а также визуализировать архитектуру разрабатываемой системы. Это значительно упростило процесс анализа функциональности, выделения ключевых компонентов и их взаимодействий для дальнейшей разработки.

Также, я сразу разработал прототип пользовательского интерфейса, он стал важным инструментом в определении логики взаимодействия пользователя с системой. Благодаря ему удалось сформировать представление о сценариях использования системы.

* 1. Дизайн-документ

В качестве основы архитектурного проектирования использовался документ ML System Design Doc, предложенный командой **Reliable ML** [1]. Он был сильно адаптирован под специфику создаваемого приложения и стал основным ориентиром при проектировании клиент-серверной архитектуры. Документ помог разбить разработку системы на этапы, а также сформулировать задачи, которые необходимо выполнить для успешной реализации проекта.

* 1. Постановка задачи

Необходимо создать прототип системы с минимально необходимым функционалом (Minimum Viable Product), который в дальнейшем мог бы быть масштабирован и дополнен новыми функциями.

Разработка системы разделена на несколько этапов:

1. Разработка серверной части. Реализация API с поддержкой авторизации, загрузки пользовательских файлов и построения индексации. В этот же этап входила разработка RAG-модуля для обработки пользовательских запросов, а также набор модульных тестов для отдельных классов и функций.
2. Реализация клиентского приложения. Разработка интерфейса на основе решений ChatPDF [ссылка] и ChatUI [ссылка] с поддержкой диалога с моделью и возможностью загрузки PDF-документов.
3. **Интеграция и тестирование.** Проверка корректности взаимодействия между клиентом и сервером, функциональное и нагрузочное тестирование.
4. **Пилотный запуск.** Развертывание системы на ограниченной пользовательской выборке, сбор обратной связи, выявление точек роста и недоработок.
   1. Архитектура серверной части

Для серверной части необходимо реализовать эндпоинты и модули для их функционирования. При проектировании компонентов серверной части я использовал UML-нотацию (см. Рисунок 4). Она позволила визуализировать основные компоненты системы и отразить из взаимодействие друг с другом и c базой данных.



Рисунок 4 – Диаграмма компонентов серверной части

* + 1. Хранение данных

Для хранения информации о пользователях я решил использовать SQLite с библиотекой SQLAlchemy. Такой выбор обусловлен необходимостью быстрых прототипирования и масштабируемости системы в будущем. SQLite идеально подходит для проекта на начальной стадии, так как не требует развертывания отдельного сервиса. Использование SQLAlchemy позволяет эффективно взаимодействовать с базой данных и упрощает расширение системы при необходимости. В будущем можно будет перейти на более сложное решение, если нагрузка на базу данных увеличится.

Для хранения файлов на этапе пилотной версии было принято решение использовать локальное файловое хранилище. Каждый документ сохраняется в виде обычного файла на сервере. Такое решение позволяет сэкономить время на интеграцию с облачными хранилищами (например, S3), и подходит для пилотной версии с ограниченной нагрузкой.

Для реализации полнотекстового поиска по фрагментам текста я решил использовать модуль FTS5 для SQLite. Это решение подходит для требований проекта и позволяет реализовать полнотекстовый поиск без необходимости интегрировать более сложные системы на начальной стадии разработки. В будущем, если потребности в поиске будут увеличиваться, можно рассмотреть переход на более мощные решения.

* 1. Клиент

В качестве основы для реализации клиентской части я решил использовать ChatUI [ссылка]. Он представляет собой современный чат-интерфейс для общения с LLM. Для поддержки просмотра PDF-файлов, необходимо встроить в приложение компонент для просмотра. Кроме того, необходима реализация клиентской логики для загрузки файлов.

* 1. Пилотный запуск

Пилотный запуск направлен на апробацию системы в условиях, приближенных к реальному использованию. Основной целью пилота является сбор обратной связи от пользователей, чтобы выявить недочеты, узкие места в логике работы, а также сформулировать направления для дальнейшего развития приложения.

* 1. Требования к работе системы

С учетом ограниченного числа пользователей пилотной версии (до 10 человек), системные требования сформулированы с учетом запаса прочности:

Пропускная способность. Исходя из предположения об 1 запросе на пользователя каждые 15 секунд, расчетная нагрузка составит:

(1)

где RPS – количество запросов за секунду. Для обеспечения устойчивой работы системы установлена целевая пропускная способность в **1.5 RPS**. Это позволит учесть различные сценарии, такие как задержки в сети или повышение активности пользователей.

Задержка ответа. Ожидаемое время отклика API — не более **1 секунды**, что обеспечит комфортную работу с системой в режиме реального времени.

Вычислительные ресурсы:

GPU: для запуска эмбеддинг модели deepvk/USER-bge-m3 (359 млн параметров) требуется ~1.44 ГБ VRAM. При пакетной обработке 32-х фрагментов до 4 ГБ VRAM.

CPU: для системы на 10 пользователей будет достаточно 2-х виртуальных ядра на запрос, таким образом, минимальные требования для системы с запасом 4 ядра CPU.

RAM: достаточно 2–4 ГБ для хранения данных и работы API.

Дисковое пространство: для хранения файлов пользователей будет достаточно дискового пространства в 200мб на каждого пользователя. Кроме хранения загруженных файлов необходимо хранить еще и индексированные чанки для каждого документа, а также базу данных для информации о пользователях и другие вспомогательные файлы. Таким образом, для поддержания системы из 10 пользователей понадобится:

(2)

* + 1. Механизмы безопасности

Для обеспечения безопасности доступа к системе в рамках пилотного проекта используется метод предопределенного ключа API, который ограничивает доступ только для заранее определенной группы пользователей. Этот ключ передается в заголовке каждого запроса, и позволяет убедиться, что доступ к системе получают только авторизованные лица.

* 1. UML-диаграммы
     1. Диаграмма вариантов использования (Use Case)

Диаграмма вариантов использования позволали выделить ключевые сценарии взаимодействия двух типов пользователей: **Пользователя** (задаёт вопросы, читает ответы, просматривает документы) и **Администратора** (кроме базового функционала, обладает правами для управления системой) (см. Рисунок 3).

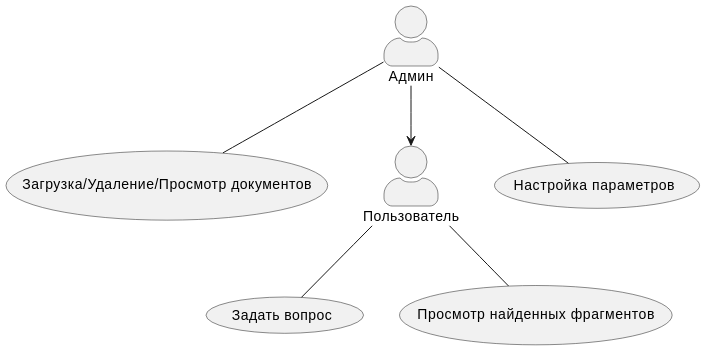


Рисунок 3 – Диаграмма вариантов использования

* + 1. Диаграмма последовательностей (Main Sequence)

Также я разработал диаграмму последовательностей, которая иллюстрирует, как объекты системы взаимодействуют друг с другом для выполнения разных сценариев работы. Эта диаграмма показывает последовательность шагов, включая отправку запроса, обработку его на сервере и возврат результатов пользователю. На основе диаграммы можно проследить, как данные перемещаются через систему и как различные компоненты обмениваются сообщениями. Этот этап помог уточнить логику взаимодействий между элементами сервиса.



* + 1. Дополнение: Диаграмма активностей (User Activity)

В дополнение к созданным диаграммам также была составлена диаграмма активностей. Это анализ активностей пользователей, который помог понять, как именно пользователи взаимодействуют с системой в рамках различных сценариев.



1. РЕАЛИЗАЦИЯ ПРОТОТИПА

Завершив этап проектирования, я занялся воплощением этих проектных решений в виде программного прототипа. В данной главе приводится описание реализованной системы, охватывающее как серверную, так и клиентскую часть, структуру хранения данных, а также модуль Retrieval-Augmented-Generation (RAG), играющий ключевую роль в моей системе.

Реализация была выполнена с учетом предварительно сформированных требований и проектных ограничений, описанных в предыдущей главе. Проект прошел несколько итераций разработки и тестирования, в процессе которых я узнавал новые для себя подходы и технологии в разработке клиент-серверных приложений. Я ориентировался на стандарты разработки подобных систем, подражая продуктовым проектам. Некоторые стандарты не были очевидны для меня в процессе проектирования, поэтому итоговая архитектура и используемые технологии незначительно отличаются от тех, что были описаны в главе про проектирование системы. В частности, в процессе изучения кодовой базы и технологий других проектов, я принял следующие архитектурные и программные решения:

1. Retrieval этап я реализовал на основе векторного поиска вместо двухступенчатого с применением BM25.
2. Использование базы данных PostgreSQL вместо SQLite. Я встретил сразу ряд сложностей при попытке реализации векторного/полнотекстового поиска с помощью SQLite, поэтому СУБД PostgreSQL послужила универсальным вариантом, у которого есть хорошая документация и большая поддержка пользователей.
3. Авторизацию с помощью предопределенных API-ключей я заменил на авторизацию с помощью JWT-токенов, т.к. это соответствует более высоким стандартам разработки.
4. Клиентская часть была написана с нуля, так как это оказалось менее трудозатратным решением по сравнению с адаптацией кодовой базы ChatUI [ссылка] под свои требования. Тем не менее, стилистическое оформление я частично позаимствовал у вышеописанного фреймворка.
5. Тестирование системы оказалось нетривиальной задачей, т.к. требовалось реализовать тестирование отдельных компонентов, в частности модуля для общения с СУБД в изолированной среде с “чистой” базой данных. Я нашел решение в использовании библиотеки testcontainers [ссылка]. Она позволяет создавать каждый раз контейнеры с новой базой данных “налету” прямо в коде проекта.
   1. Общая структура системы

Разработанная система состоит из двух основных частей: клиентского интерфейса и серверного приложения. Между ними осуществляется обмен данными по протоколу HTTP, а также через потоковую передачу данных посредством Server-Sent Events (SSE). Ключевым элементом архитектуры является модуль RAG, обеспечивающий возможность семантического поиска и генерации ответов на естественном языке на основе загруженных документов.

* 1. Серверная часть

Серверная часть реализована на языке Python с использованием асинхронного фреймворка FastAPI. Для работы с базой данных используется библиотека SQLAlchemy. Основные функции серверной части включают:

1. Обработку HTTP- и SSE-запросов от клиента.
2. Взаимодействие с RAG-системой для генерации ответов.
3. Аутентификация и авторизация пользователей на основе JWT-токенов.
4. Асинхронный доступ к хранимым файлам и метаданным.
5. Поддержка многопользовательского режима.

Благодаря асинхронной архитектуре, сервер способен эффективно обрабатывать множество одновременных клиентов.

* 1. Итоговая архитектура

Для формализации архитектуры системы была выбрана C4-нотация. Она предлагает гибкий набор инструментов для проектирования программных систем, и включает в себя несколько уровней детализации:

1. Контекст (Context) – показывает внешние системы и пользователей.
2. Контейнеры (Containers) – архитектура приложения без глубокого погружения в техническую часть. Она отображает основные логические блоки и используемые технологии.
3. Компоненты (Components) – раскрывает архитектуру отдельных контейнеров.
4. Код (Code) – самый низкий уровень абстракции, чтобы показать классы и их связи.

Для проекта я использовал второй и третий уровни – “Контейнеры” и “Компоненты”, они позволили описать систему в достаточной детализации без избыточных подробностей о кодовой базе проекта.

На Рисунке 1 представлена диаграмма контейнеров, показывающая общее взаимодействие клиента, сервера, базы данных и RAG-пайплайна.



Рисунок 1 – диаграмма C4 (уровень контейнеров).

Следующий уровень детализации – компоненты серверной части (Рисунок 2). На нем я выделил ключевые модули:

1. API-сервис – точка входа, обрабатывает запросы от клиента, авторизует пользователей, управляет загрузкой файлов, и инициирует взаимодействие с RAG-пайплайном.
2. Сервис индексации RAG – обрабатывает загруженные документы, извлекает текст, и разделяет его на фрагменты для последующего поиска.
3. Сервис генерации RAG – принимает пользовательский запрос, извлекает релевантные фрагменты, генерирует финальный ответ с помощью большой языковой модели.



Рисунок 2 – диаграмма C4 третьего уровня (Components)

Наконец, я визуализировал более детализированную схему взаимодействия внутри RAG-пайплайна, куда включил этапы индексации и генерации. (Рисунок 3).



Рисунок 3 - диаграмма C4 (уровень компонентов, RAG-пайплайн)

* 1. Клиентская часть

Клиент (Рисунок 4) реализован с помощью библиотеки React в виде одностраничного веб-приложения (SPA). Его функционал включает:

1. Интерактивный чат с возможностью ввода и отображения истории сообщений.
2. Поддержка потоковых ответов от сервера.
3. Регистрация и авторизация пользователей.
4. Загрузка и просмотр PDF-документов.
5. Навигация по загруженным файлам.



Рисунок 4 – Пользовательский интерфейс веб-приложения

* 1. Хранение данных

В качестве основной СУБД используется PostgreSQL. В базе хранятся:

1. Информация о пользователях
2. История запросов и ответов
3. Метаданные, связанные с загруженными файлами
4. Чанки и их векторные представления

Сами PDF-файлы сохраняются в локальном хранилище. Доступ к ним происходит асинхронно по запросу клиента.

* 1. Модуль Retrieval-Augmented-Generation

Работа модуля включает два этапа: Индексация и Генерация (см. Рисунок 3). Первоначально я рассматривал классического подход с использованием поиска на основе BM25. Но несмотря на его простоту и эффективность в ряде задач, он имеет ограничение, связанное с работой с мультиязычными данными. В моей системе как запросы, так и чанки в базе данных могут быть представлены на русском или английском языках. BM25 оперирует на уровне отдельных слов, а значит не способен учитывать семантику.

Вместо этого я реализовал семантический поиск с использованием векторных представлений как для запросов, так и для чанков из базы данных. Для вычисления схожести используется косинусная мера. Это позволяет системе находить релевантные фрагменты даже с учетом различия в формулировках и языках.

1. ТЕСТИРОВАНИЕ

Для проверки корректности и надёжности функционала уже в процессе разработки я составлял модульные и интеграционные тесты с использованием библиотеки Pytest и асинхронного клиента из библиотеки HTTPX. Тестами покрыты основные компоненты системы:

1. Регистрация и авторизация пользователей.
2. Хранение и получение истории сообщений.
3. Работа с SSE, и получение потоковых ответов от системы.
4. Загрузка и парсинг PDF-файлов.
5. Интеграция с RAG-системой и корректная обработка запросов.

Тестирование проводится автоматически, что позволяет оперативно выявлять ошибки при изменении кода и вносить улучшения без снижения стабильности системы.

Кроме того, я разработал модуль, который проводит бенчмарк системы с помощью LLM-as-а-Judge подхода. Он подключается к модулю Retrieval-Augmented-Generation и выполняет вопрос-ответное тестирование на основе датасета.

* 1. Датасет

Первоначально я рассматривал датасет QASPER для тестирования системы, однако в процессе разработки я выявил его ограничения: вопросы в датасете слишком общие, к тому же каждый вопрос связан только с одной статьей. Эти ограничения делают QASPER малопригодным для тестирования сложных retrieval-механизмов. В итоге я выбрал датасет FRAMES (Factuality, Retrieval, And reasoning MEasurement Set) [ссылка]. Основные характеристики датасета:

1. 824 сложных вопросов, требующих информации из 2-15 статей из Wikipedia.
2. Широкий спектр тем: история, спорт, наука, здоровье, животные и др.
3. Для каждого вопроса представлен эталонный ответ, а также список релевантных статей из Wikipedia.

Датасет был разработан и выпущен 24 января 2025 года исследователями из Google с целью создания стандартизированного набора задач для оценки работы Retrieval-Augmented-Generation систем. В статье [ссылка] авторы демонстрируют, что даже передовые большие языковые модели, такие как Gemini-Pro-1.5, сталкиваются со значительными трудностями при ответах на вопросы из FRAMES. При этом RAG-системы, использующие даже простые механизмы поиска, например BM25, показывают результаты лучше. Это подтверждает, что добавление поискового механизма может существенно улучшить качество ответов.

Для тестирования я случайным образом отобрал из датасета 50 вопросов, и проиндексировал связанные с ними статьи. Чтобы приблизить эксперимент к реальным условиям, в индекс также добавил нерелевантные статьи, содержащие информацию по другим вопросам. Этот шаг позволяет проверить, насколько эффективно система извлекает нужные данные и игнорирует лишнюю информацию. Итоговая выборка включает 330 статей и 50 вопросов.

* 1. Конфигурации

Для сравнительного тестирования я использовал три разных конфигурации вопрос-ответных систем:

1. Модуль RAG из моей системы.
2. Версия Agentic RAG. Для его реализации я использовал библиотеку smolagents [ссылка], а в качестве инструмента подключил поисковой механизм моей системы. В результате получилась система, способная пошагово рассуждать и извлекать данные, если нужно. Цепочка рассуждений ограничена 6 шагами.
3. LLM без поиска.

Во всех конфигурациях использовалась большая языковая модель openai/gpt-4o-mini [ссылка] и эмбеддинг-модель deepvk/USER-bge-m3 [ссылка].

* 1. Анализ результатов

Стоит отметить, что большие языковые модели склонны к более лояльной оценке своих текстов [9], поэтому для метрик LLM-as-Judge я использовал другую модель: google/gemini-pro [5] и библиотеку RAGAS.

* + 1. Сравнение метрик на разных конфигурациях

|  |  |  |  |
| --- | --- | --- | --- |
|  | RAG | Agentic-RAG | LLM |
| Faithfulness | 0.54 | 0.08 | 0.25 |
| Response Relevancy | 0.29 | 0.6 | - |
| Answer Correctness | 0.33 | 0.49 | 0.53 |
| BERTScore | 0.8 | 0.82 | 0.81 |
| Среднее время работы (с) | 6.5 | 29 | 2.37 |

* + 1. Выводы по метрикам

RAG требует доработки алгоритма чанкинга для улучшения качества поиска и общего качества извлекаемых чанков. Нужно исследовать эвристические подходы к формированию чанков и сравнить с текущим семантическим подходом.

Agentic-RAG показывает лучшие результаты по извлечению фрагментов благодаря множественным вызовам поиска по базе знаний. Однако необходимо улучшить формулировку промежуточных запросов для повышения их релевантности и точности, также переработать запрос для формирования финального ответа.

* 1. Пилотное тестирование

В рамках разработки проекта я также провел небольшое пилотное тестирование. Для этого я развернул приложение на сервере и открыл доступ к приложению через Интернет. Целью тестирования было получение обратной связи о работоспособности интерфейса, корректности генерации ответов и общего впечатления пользователей о работе приложения. Кроме того, было важно узнать, насколько потенциально полезным могло бы быть такое приложение в учебном процессе.

Я сформировал 6 вопросов и внес их в сервис Google Forms для автоматического сбора ответов пользователей. В результате пользователи получили в свободное использование приложение и анкеты для заполнения (см. Приложение 1).

В результате опроса я пришел к следующим выводам:

1. Система корректно обрабатывала запросы, но в некоторых ситуациях давала слишком общие ответы.
2. Пользователи положительно оценили концепт дизайна интерфейса. Особо отметили возможность просмотра PDF-файлов прямо в приложении.

Результаты тестирования были мной учтены при доработке приложения.

ЗАКЛЮЧЕНИЕ

Современная образовательная сфера переживает масштабную трансформацию, связанную с искусственным интеллектом. Особую роль в этих изменениях играют большие языковые модели, в частности чат-боты на их основе. Большие языковые модели предоставляют быстрые и понятные ответы на вопросы, что делает их удобным инструментом при изучении материалов: пользователю гораздо проще задать вопрос модели, чем искать ответы в литературе. Несмотря на их удобство, модели склонны к галлюцинациям, особенно в специфических областях знаний, что негативно сказывается на процессе обучения.

Для преодоления этой проблемы разработчики и исследователи применяют подход Retrieval Augmented Generation – он позволяет значительно улучшить качество ответов большой языковой модели, подключив к ней внешние источники знаний. Популярность RAG-систем в образовательной сфере стремительно растет. Согласно отчету компании Яндекс “ИИ и высшее образование” [ссылка] решения на базе искусственного интеллекта приобретают все большую популярность среди вузов России. Инструменты на базе искусственного интеллекта становятся не заменой человеку, а его интеллектуальным помощником, позволяя повышать эффективность учебного процесса. Предполагается, что RAG позволит работать образовательным организациям с заранее верифицированным контентом. Необходимые материалы могут быть собраны и валидированны самой образовательной организацией, а затем загружены в специальную библиотеку, которая подключается к большой языковой модели.

В рамках выпускной квалификационной работы я спроектировал и реализовал гибкую масштабируемую систему, использующую RAG. Ее главная особенность заключается в способности налету индексировать данные и выдавать ответ пользователю в реальном времени. Из-за такой особенности появились высокие требования к производительности компонентов, особенно к процессу индексации. Несмотря на ограничения, мне удалось реализовать прототип, который демонстрирует практическую применимость подобного приложения. Кроме того, разработанная система модульная и легко адаптируется под разные сценарии использования: от поддержки учебного процесса до работы с внутренними корпоративными документами. Она легко расширяется дополнительными инструментами и адаптируется под конкретные задачи. Проведенное тестирование показало надежность и стабильность системы. Реализованное приложение соответствует современным стандартам разработки и может служить основой для дальнейшей разработки прикладных и коммерческих продуктов в сфере образования.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Lewis P. et al. Retrieval-augmented generation for knowledge-intensive nlp tasks //Advances in neural information processing systems. – 2020. – Т. 33. – С. 9459-9474.
2. Barnett S. et al. Seven failure points when engineering a retrieval augmented generation system //Proceedings of the IEEE/ACM 3rd International Conference on AI Engineering-Software Engineering for AI. – 2024. – С. 194-199.
3. Ma X. et al. Query rewriting in retrieval-augmented large language models //Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing. – 2023. – С. 5303-5315.
4. Yu H. et al. Evaluation of retrieval-augmented generation: A survey //CCF Conference on Big Data. – Singapore : Springer Nature Singapore, 2024. – С. 102-120.
5. Glass M. et al. Re2G: Retrieve, rerank, generate //arXiv preprint arXiv:2207.06300. – 2022.
6. Es S. et al. Ragas: Automated evaluation of retrieval augmented generation //Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations. – 2024. – С. 150-158.
7. Wang X. et al. Searching for best practices in retrieval-augmented generation //arXiv preprint arXiv:2407.01219. – 2024.
8. Hu Y., Lu Y. Rag and rau: A survey on retrieval-augmented language model in natural language processing //arXiv preprint arXiv:2404.19543. – 2024.
9. Yan S. Q. et al. Corrective retrieval augmented generation. – 2024.
10. Wei J. et al. Chain-of-thought prompting elicits reasoning in large language models //Advances in neural information processing systems. – 2022. – Т. 35. – С. 24824-24837.
11. Asai A. et al. Self-rag: Learning to retrieve, generate, and critique through self-reflection //The Twelfth International Conference on Learning Representations. – 2023.
12. Levels Of Text Splitting [Электронный ресурс]. — 2024. — Режим доступа: https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5\_Levels\_Of\_Text\_Splitting.ipynb (дата обращения: 09.12.2024).
13. Sarthi P. et al. Raptor: Recursive abstractive processing for tree-organized retrieval //The Twelfth International Conference on Learning Representations. – 2024.
14. Dasigi P. et al. A dataset of information-seeking questions and answers anchored in research papers //arXiv preprint arXiv:2105.03011. – 2021.
15. ML System Design Doc – Reliable ML [Электронный ресурс]. — 2025. — Режим доступа: https://github.com/IrinaGoloshchapova/ml\_system\_design\_doc\_ru/tree/main (дата обращения: 15.03.2025).
16. HuggingFace [Электронный ресурс]. — 2025. — Режим доступа: https://huggingface.co/deepvk/USER-bge-m3 (дата обращения: 15.03.2025).
17. Zhang T. et al. Bertscore: Evaluating text generation with bert //arXiv preprint arXiv:1904.09675. – 2019.
18. Gu J. et al. A survey on llm-as-a-judge //arXiv preprint arXiv:2411.15594. – 2024.
19. Krishna S. et al. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation //arXiv preprint arXiv:2409.12941. – 2024.
20. VseGpt – провайдер для больших языковых моделей [Электронный ресурс]. — 2025. — Режим доступа: https://vsegpt.ru/Docs/Models (дата обращения: 26.02.2025).
21. Robertson S. E. et al. Okapi at TREC-3 //Nist Special Publication Sp. – 1995. – Т. 109. – С. 109.
22. Xu W. et al. Pride and prejudice: LLM amplifies self-bias in self-refinement //arXiv preprint arXiv:2402.11436. – 2024.
23. Open-source codebase powering the HuggingChat app [Электронный ресурс]. — 2025. — Режим доступа: https://github.com/huggingface/chat-ui (дата обращения: 26.02.2025).