СОДЕРЖАНИЕ

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.

Эмбеддинг-модель (энкодер, эмбеддер) – модель, преобразующая входные данные (текстовые) в векторное представление (эмбеддинги).

Промпт – запрос в большую языковую модель.

Пайплайн (конвейер) – последовательность действий или процессов преобразований.

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

RAG (Retrieval Augmented Generation)– подход, при котором большая языковая модель генерирует ответы на основе извлеченной из внешних источников информации

Retrieve (получать, извлекать) и Generate (генерировать) – основные компоненты RAG-пайплайна.

Чанк (фрагмент текста) – извлеченный на этапе Retrieve фрагмент информации из внешнего источника.

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) [1], 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.

Интересно, что идея разговора с книгой не новая: так, еще в фантастическом романе “Алмазный век” [2] писателя Нила Стивенсона есть описание “Букваря благородных девиц” – уникальной интерактивной книги, с которой можно было разговаривать, задавать ей вопросы. Такой “диалог” с книгой отражает потребность человека в том, чтобы не только прочитать, но и обсудить материал. Современные технологии, в частности RAG, позволяют воплотить в реальность образ книги из романа в виде обучающего материала, с которым можно взаимодействовать в формате беседы.

Целью выпускной квалификационной работы является разработка системы, использующей подход RAG для помощи пользователям в изучении учебных материалов, в частности научных статей в формате PDF. Система должна обеспечивать автоматический поиск и извлечение релевантной информации из предоставленного документа, перевод с английского языка, а также отвечать 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 RAG, 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 RAG и обеспечить взаимодействие с пользователем в реальном времени.

4 Реализовать алгоритм оценки качества системы с точки зрения полноты и точности предоставляемых ответов.

1 ТЕОРЕТИЧЕСКИЕ ОСНОВЫ RETRIEVAL AUGMENTED GENERATION

С развитием технологий обработки естественного языка большие языковые модели показали 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, в результате функционируя как неявная база знаний. Однако такая архитектура имеет существенные ограничения. [1,3]

Во-первых, языковые модели обучаются на статических наборах данных и не могут обновлять свои знания без дорогостоящего дообучения. Во-вторых, 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 в 2022 году был предложен подход Retrieval Augmented Generation (RAG) [1]. Основная идея RAG заключается в том, чтобы обогатить запрос пользователя дополнительным контекстом, и только потом 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. Она включает в себя три ключевых этапа: разделение корпуса данных на фрагменты (Chunking), поиск релевантных фрагментов (Retrieve) и генерацию ответа на их основе (Generate). На первом этапе система извлекает из внешнего хранилища фрагменты, наиболее подходящие под запрос. Затем фрагменты вместе с пользовательским запросом подаются на вход большой языковой модели, которая генерирует финальный ответ, используя как внутренние знания, так и предоставленный контекст.

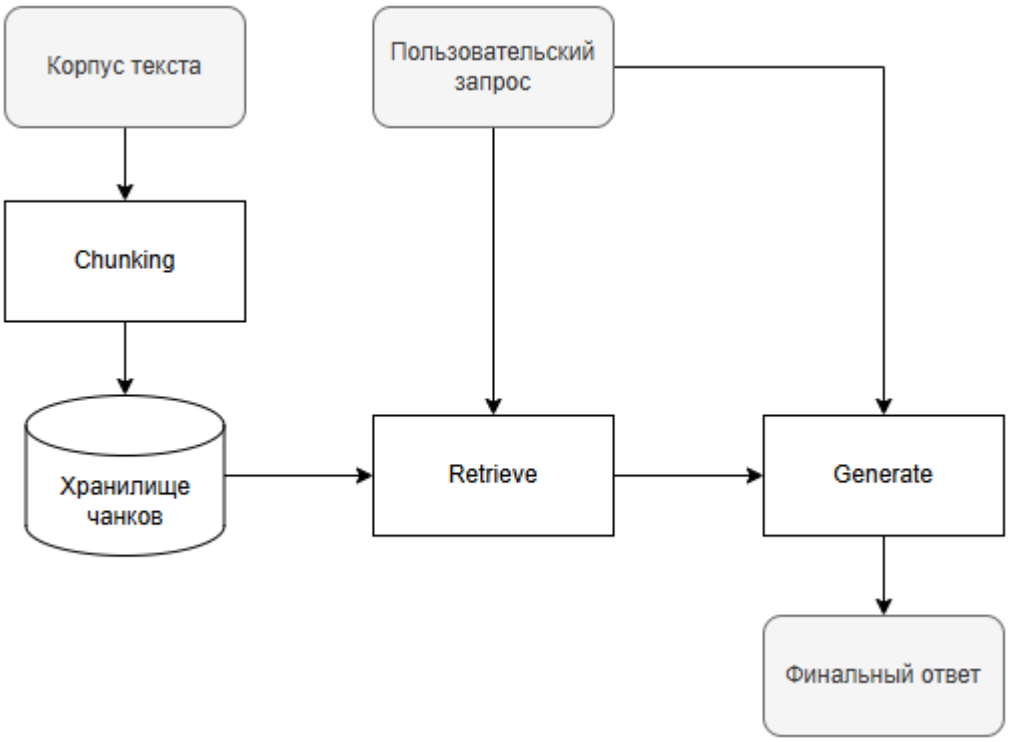


Рисунок 1 - Устройство RAG-пайплайна

1.1 Ключевые этапы RAG-пайплайна

Подход RAG модульный и состоит из нескольких этапов, каждый из которых играет важную роль в обеспечении качества итогового ответа.

1.1.1 Chunking

Этап Chunking (разбиение текста на фрагменты) представляет собой подготовку корпуса текста для поиска. Исходные документы разбиваются на небольшие логически связанные чанки. Задача этого этапа получить фрагменты, которые содержат достаточное количество информации для понимания контекста. Кроме того, они не должны быть слишком длинными, чтобы поместиться в контекст большой языковой модели [3, 4]. 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. [3]

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.

1.1.2 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 [5]. Эти методы обеспечивают быструю обработку запросов, но они не учитывают семантику текста, из-за чего показывают низкую точность при наличии переформулировок или опечаток.

2 Методы на основе эмбеддингов. В этом случае 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 выполнить “грубый поиск”, отобрав 100 кандидатов с помощью BM25, а затем провести их переранжирование с использованием эмбеддинг-моделей.

2 ПРОДВИНУТЫЕ ПОДХОДЫ

С момента появления RAG было предложено множество подходов для повышения его эффективности. Улучшения затрагивают практически все стадии пайплайна. Далее рассмотрим некоторые усовершенствования, сгруппированные по соответствующим этапам.

2.1 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, где плотность информации может значительно варьироваться [3]. В идеале чанк должен представлять собой логически завершённую мысль, которая не теряет смысла вне контекста. Оптимально, если чанк охватывает не отдельные предложения, а их логически связанные группы, так как нарушение границ предложений или смысловых блоков может привести к потере критически важной информации.

2.1.1 Семантический Chunking

1 Similarity Chunking [6]. Данный метод использует эмбеддинг-модели для группировки связанных по смыслу предложений. В зависимости от имплементаций алгоритм может меняться. В общем виде он реализуется в два этапа. Первым этапом текст делят на предложения и вычисляют их векторные представления. Затем между соседними парами предложений вычисляют косинусные расстояния, в результате чего получается распределение значений (см. Рисунок 2). Наконец, текст разделяется по пороговому значению, например, выбирается 95-й процентиль по получившемуся распределению (красная линия на Рисунке 2).

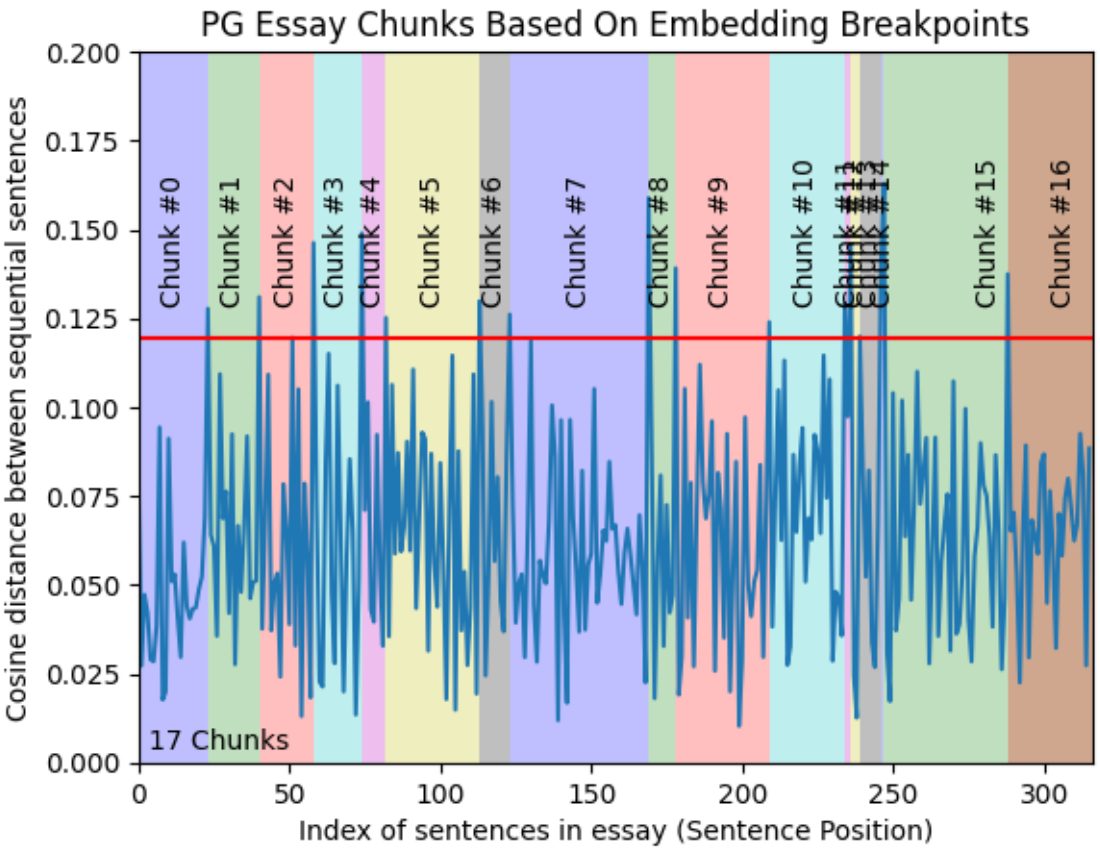


Рисунок 2 – Распределение косинусных расстояний

2 Perplexity Chunking. [7] 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) [7] для каждого предложения x_i на основе предшествующих предложений:

$$PPL_M(x_i) = \frac{\sum_{k=1}^{K} PPL_M\left(t_k^i \middle| t_{<k}^i,\ t_{<i}\right)}{K} \quad (1)$$

где K - общее количество токенов в x_i, t_i^k - k-й токен в x_i, а $t_{<i}$ обозначает все токены, предшествующие x_i. Для определения границ чанков алгоритм анализирует следующую последовательность:

$$PPL_{seq} = (PPL_M(x_1), PPL_M(x_2), \ldots, PPL_M(x_n)) \quad (2)$$

В поисках минимальных значений. Минимумы рассматриваются как потенциальные границы фрагментов, если верно хотя бы одно из следующих утверждений:

1 Значения PPL по обе стороны точки выше, чем в самой точке, и разница хотя бы с одной стороны превышает заданный порог θ:

$$\left\{i \mid min\left(PPL_M(x_{i-1}), PPL_M(x_{i+1})\right) - PPL_M(x_i) > \theta\right\} \quad (3)$$

2 Разница между левой точкой и текущей больше θ, а значение справа равно значению в текущей точке:

$$\{i \mid PPL_M(x_{i-1}) - PPL_M(x_i) > \theta \; and \; PPL_M(x_{i+1}) = PPL_M(x_i)\} \quad (4)$$

2.2 Rewriting

Цель этапа Rewriting заключается в адаптации пользовательского запроса к структуре и стилю данных, хранящихся в индексе [8]. Это особенно важно, если пользователь использует в своем запросе общие или неоднозначные формулировки, которые не соответствуют документам в целевом корпусе.

2.2.1 HyDE

Метод HyDE (Hypothetical Document Expansion) [9] 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 фактологические 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.

2.3 Reranking

После первоначального извлечения релевантных документов на этапе Retrieval (например, с помощью алгоритма BM25 [5]), часто применяются алгоритмы переранжирования (этап Reranking), уточняют порядок релевантности результатов. Рассмотрим два наиболее распространенных подхода:

1 Bi-Encoder [10, 11]. Запрос и каждый документ обрабатываются раздельно с помощью эмбеддинг моделей (энкодеров на Рисунке 3). На выходе получаются векторы, между которыми вычисляется мера векторной близости. На Рисунке 3 показано, как фрагменты текста преобразуются в векторы для вычисления их семантической схожести.

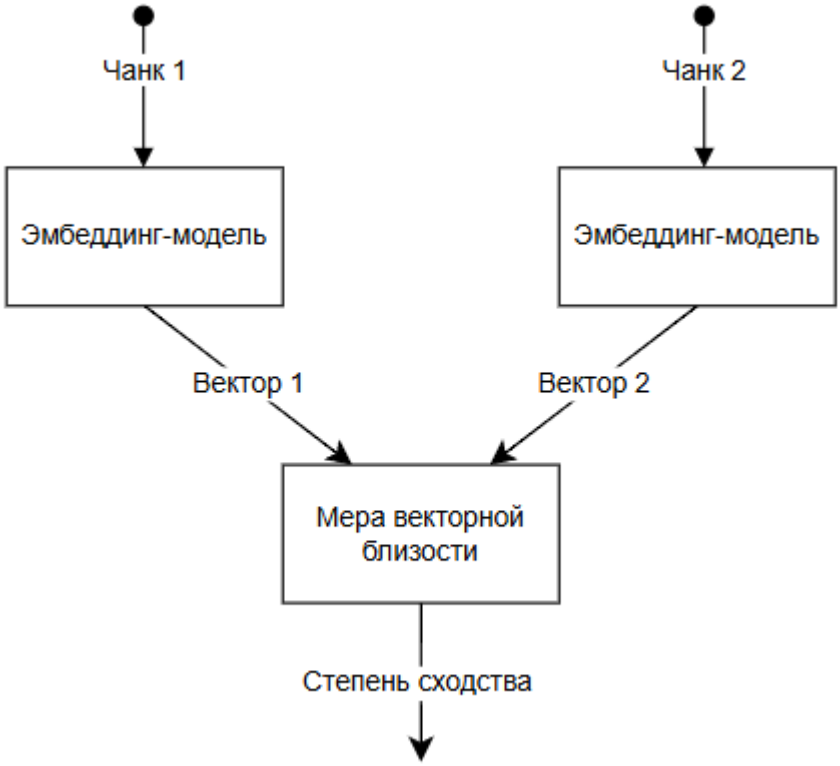


Рисунок 3 — Bi-Encoder

2 Cross-Encoder [10, 11]. Запрос и документ объединяются в одну строку и подаются на вход эмбеддинг модели. Полученный вектор подается на вход предобученного классификатора, который возвращает значение сходства для фрагментов (см. Рисунок 4).

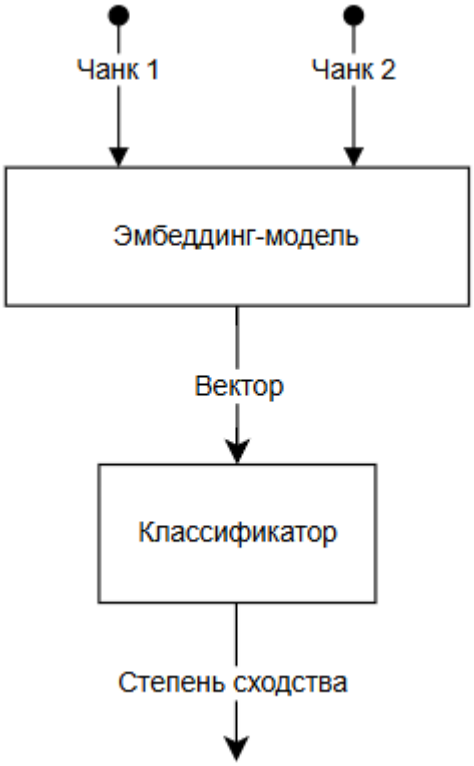


Рисунок 4 — Cross-Encoder

В статье “In defense of cross-encoders for zero-shot retrieval” [12] приводятся сравнение подходов Bi-Encoder и Cross-Encoder, где Cross-Encoder показывает себя точнее в задаче реранжирования. Однако засчет своей архитектуры Cross-Encoder существенно более вычислительно затратный: для его работы необходимо попарное вычисление степеиь сходства между индексированными чанками и запросом, сложность достигает $O(n^2)$. Bi-Encoder в свою очередь позволяет предварительно индексировать все чанки, а затем проводить их сравнения с вектором запроса, таким образом сложность достигает O(n). Из-за такого различия в вычислительных затратах Cross-Encoder подходит хуже для задач в реальном времени.

2.4 Agentic RAG

Agentic RAG [12] представляет собой усовершенствованный подход к построению RAG. 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 [17]. 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 [13]. 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 цепочка рассуждений работает циклически до достижения финального ответа, в рамках одного цикла выполняются следующие шаги:

1 Thought (рассуждение): модель генерирует шаг рассуждения и решает, какое действие необходимо вызвать и с какой целью.

2 Action (действие): модель на основе шага рассуждения самостоятельно вызывает инструмент, подставляя необходимые параметры, либо инициирует генерацию финального ответа.

3 Observation (наблюдение, вывод) Получив результаты инструмента, модель делает выводы, и цикл повторяется (см. Рисунок 5).

Question	Musician and satirist Allie Goertz wrote a song about the "The Simpsons" character Milhouse, who Matt Groening named after who?
Thought 1	The question simplifies to "The Simpsons" character Milhouse is named after who. I only need to search Milhouse and find who it is named after.
Action 1	Search[Milhouse]
Observation 1	Milhouse Mussolini Van Houten is a recurring character in the Fox animated television series The Simpsons voiced by Pamela Hayden and created by Matt Groening.
Thought 2	The paragraph does not tell who Milhouse is named after, maybe I can look up "named after".
Action 2	Lookup[named after]
Observation 2	(Result 1 / 1) Milhouse was named after U.S. president Richard Nixon, whose middle name was Milhous.
Thought 3	Milhouse was named after U.S. president Richard Nixon, so the answer is Richard Nixon.
Action 3	Finish[Richard Nixon]

Рисунок 5 – пример цепочки рассуждений ReAct

Для реализации агентных систем такого рода можно использовать библиотеку smolagent [16]. Она предоставляет возможность задавать

инструменты для модели, а внутри реализует алгоритм, подобный ReAct: он начинается с пользовательского запроса, а далее в цикле происходит цепочка размышлений-действий (см. Рисунок 6).

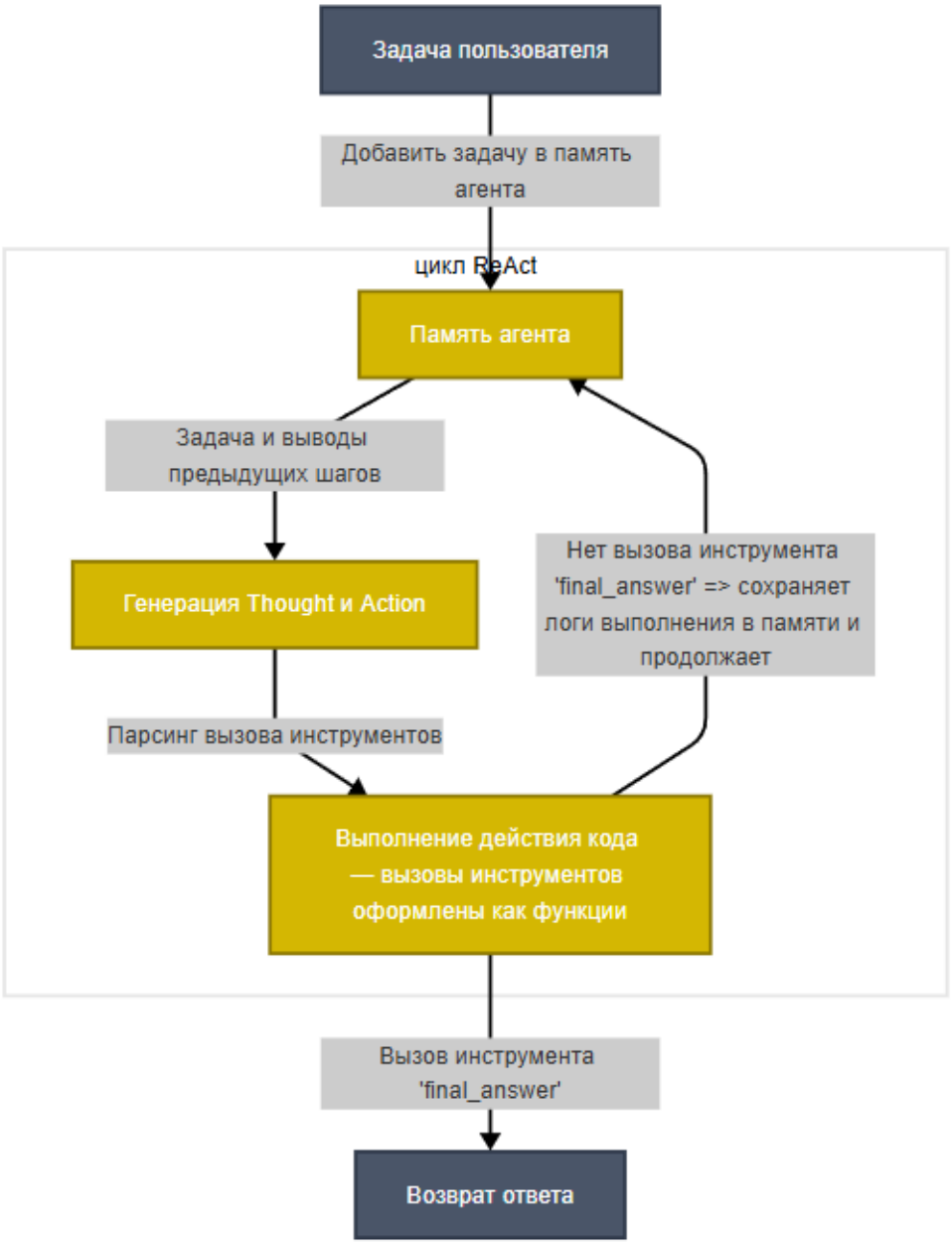


Рисунок 6 – Цикл выполнения smolagents

3 ТОЧКИ ОТКАЗА СИСТЕМ С RETRIEVAL AUGMENTED GENERATION

RAG-системы 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 ответов. В статье “Seven Failure Points When Engineering a Retrieval Augmented Generation System” [3] 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.

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 или ошибками в алгоритмах поиска или реранжирования.

3 Ошибки при добавлении чанков в запрос к модели. Релевантные документы быть найдены, но не попали в финальный промпт из-за ограничений контекста модели или неправильной предобработки перед добавлением в финальный промпт.

4 Потеря информации во время разделения текста на чанки: необходимая для ответа информация содержится в корпусе текста, но в процессе индексации получаются противоречивые или бессмысленные чанки.

5 Неправильный формат ответа модели. Эта ошибка возникает в случае, если модель игнорирует инструкцию возвращать ответ в определенном формате, например в виде таблицы или словаря.

6 Неправильная степень специфичности. Ответы, формируемые моделью, могут быть либо чрезмерно обобщёнными, либо излишне

детализированными. Это снижает их полезность и не соответствует ожиданиям пользователя.

7 Неполные ответы: система может предоставлять ответы, содержащие лишь часть необходимой информации, даже если информация для полного ответа доступна в найденных документах.

Кроме того, RAG-системы подвержены деградации качества со временем и требуют периодической переоценки ключевых компонентов. Особенно это критично при изменении пользовательских предпочтений или обновлении источников данных. [3]

4 ОЦЕНКА СИСТЕМ С RETRIEVAL AUGMENTED GENERATION

Как было показано в предыдущей главе, при проектировании и внедрении систем с RAG 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” [14] авторы помогают структурировать метрики для оценки RAG по данным, на основе которых высчитываются метрики. На этапе поиска (Retrieve):

1 Релевантность (найденные чанки ↔ запрос пользователя) – определяет релевантность найденных чанков к запросу пользователя.

2 Точность (найденные чанки ↔ эталонный набор чанков) – определяет, насколько точно система отбирает чанки относительно эталонных чанков, определенных в используемом датасете.

На этапе генерации (Generate):

1 Релевантность (ответ ↔ запрос пользователя) – определяет степень соответствия ответа системы пользовательскому запросу по теме, и требованиям пользователя

2 Достоверность (ответ ↔ найденные чанки) – вычисляет степень достоверности ответа относительно найденных ответов

3 Правильность (ответ ↔ эталонный ответ) – вычисляет степень соответствия ответа системы эталонному ответу, представленному в датасете.

Библиотеки и датасеты предоставляют воспроизводимые, масштабируемые и автоматизированные способы оценки, один из таких фреймворков RAGAS.

Библиотека RAGAS [15] разработана специально для оценки качества работы RAG-систем. Она 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 [18]: большая языковая модель используется в качестве судьи. Её задача заключается в оценке качества работы системы. Далее рассмотрим некоторые из этих метрик более подробно.

4.1.1 Релевантность поиска

Метрика релевантности поиска (Context Precision в RAGAS) рассчитывается, как доля релевантных к запросу чанков. В приведенной ниже формуле 3 рассчитывается данная метрика:

$$Context\ Precision@K\ = \frac{\sum_{k=1}^{K}(Precision@k\ \times\ u_k)}{\text{Количество релевантных чанков}} \quad (3)$$

где K – количество найденных чанков, $u_k \in \{0, 1\}$ – значение релевантности для найденного чанка на позиции k, определенное с помощью большой языковой модели.

$$Precision@k\ = \frac{true\ positives@k}{(true\ positives@k\ +\ false\ positives@k)} \quad (3)$$

4.1.2 Достоверность генерации

Метрика достоверности (Faithfulness) измеряет, насколько фактологически 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:

$$\text{Faithfulness Score} = \frac{\text{Количество релевантных утверждений}}{\text{Общее количество утверждений}} \quad (4)$$

4.1.3 Релевантность генерации

Метрика релевантности ответа (Response Relevancy) 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.

4.1.4 Правильность генерации

Метрика корректности ответа (Answer Correctness) оценивает соответствие ответа эталонному. Балл варьируется от 0 до 1, где 1 означает полное совпадение. Алгоритм расчета метрики:

1 Оценка фактологической корректности: измеряет степень совпадения фактов между сгенерированным ответом и эталонным. Рассчитывается, как F1-Score:

$$F1\ Score = \frac{2 \times TP}{2 \times TP + FP + FN} \quad (5)$$

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.

2 Оценка семантического сходства: рассчитывается на основе косинусной близости эмбеддингов сгенерированного и эталонного ответов.

3 Ответ: рассчитывается, как взвешенное среднее между фактологическим соответствием и семантическим сходством.

5 ПРОЕКТИРОВАНИЕ СИСТЕМЫ

Прежде чем начинать разработку проекта, был проведен этап проектирования архитектуры. На данном этапе необходимо было уточнить требования к функционалу приложения и инструментам разработки.

Для проектирования были использованы средства визуального моделирования, в частности UML-диаграммы. Они позволили формализовать и структурировать требования к системе на раннем этапе, а также визуализировать архитектуру. Это позволило выделить ключевые компоненты и их взаимодействие, что упростило дальнейшую разработку приложения.

В дополнение был разработан прототип пользовательского интерфейса. Он стал важным инструментом в определении логики взаимодействия пользователя с системой. Благодаря ему удалось сформировать представление о сценариях использования системы.

5.1 Дизайн-документ

В качестве основы для проектирования использовался документ ML System Design Doc, предложенный командой Reliable ML [19]. 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.

5.2 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 [20] и ChatUI [21] с поддержкой диалога с большой языковой моделью и возможностью загрузки PDF-документов.

3 Интеграция и тестирование. Проверка корректности взаимодействия между клиентом и сервером, функциональное и нагрузочное тестирования.

4 Пилотный запуск. Развертывание системы на ограниченной пользовательской выборке, сбор обратной связи, выявление точек роста и недоработок.

5.3 Архитектура серверной части

Для серверной части необходимо реализовать эндпоинты и модули для их функционирования. При проектировании компонентов серверной части применялась UML-нотация. Диаграмма компонентов позволила визуализировать основные части системы и отразить их взаимодействие друг с другом и с базой данных (см. Приложение А).

5.3.1 Хранение данных

Для хранения информации о пользователях на ранних этапах проекта было решено использовать SQLite [22] с библиотекой SQLAlchemy [23]. Такой выбор обусловлен необходимостью быстрого прототипирования. 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, что позволяет сэкономить время на интеграции с облачными хранилищами и подходит для пилотной версии с ограниченной нагрузкой.

Для реализации информационного поиска по фрагментам текста я решил использовать модуль FTS5 для SQLite. Это решение подходит под требования проекта и позволяет быстро реализовать полнотекстовый поиск. В будущем, если

потребности в поиске будут увеличиваться, можно рассмотреть переход на более мощные решения.

5.4 Клиент

В качестве основы для реализации клиентской части я решил использовать ChatUI [21]. Он представляет собой современный чат-интерфейс для общения с LLM. Для поддержки чтения PDF-файлов необходимо встроить в приложение компонент для просмотра. Кроме того, необходима реализация клиентской логики для загрузки файлов.

5.5 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.

5.6 Требования к работе системы

С учетом ограниченного числа пользователей пилотной версии (до 10 человек), системные требования сформулированы с учетом запаса по производительности. Сперва была рассчитана пропускная способность. Исходя из предположения об 1 запросе на пользователя каждые 15 секунд, расчетная нагрузка составит:

$$10 \text{ пользователей} \times \frac{1 \text{ запрос}}{15 \text{ секунд}} = 0.67\ RPS \tag{6}$$

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. Ожидаемое время отклика API — не более 1 секунды, что обеспечит работу с системой в режиме реального времени. Кроме того, были рассчитаны требования для вычислительных ресурсов:

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

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

Для хранения файлов пользователей будет достаточно дискового пространства в 200 мегабайт 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 метаданных файлов. Таким образом, для поддержания системы из 10 пользователей понадобится:

$$200\text{МБ} \times 2 \times 10 + 500\text{МБ} \cong 4.5\text{ГБ} \quad (7)$$

5.6.1 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.

5.7 UML-диаграммы

Диаграмма вариантов использования позволила выделить ключевые сценарии взаимодействия двух типов пользователей: Пользователя (задаёт вопросы, читает ответы, просматривает документы) и Администратора, который кроме базового функционала обладает правами для настройки системы (см. Рисунок 8).

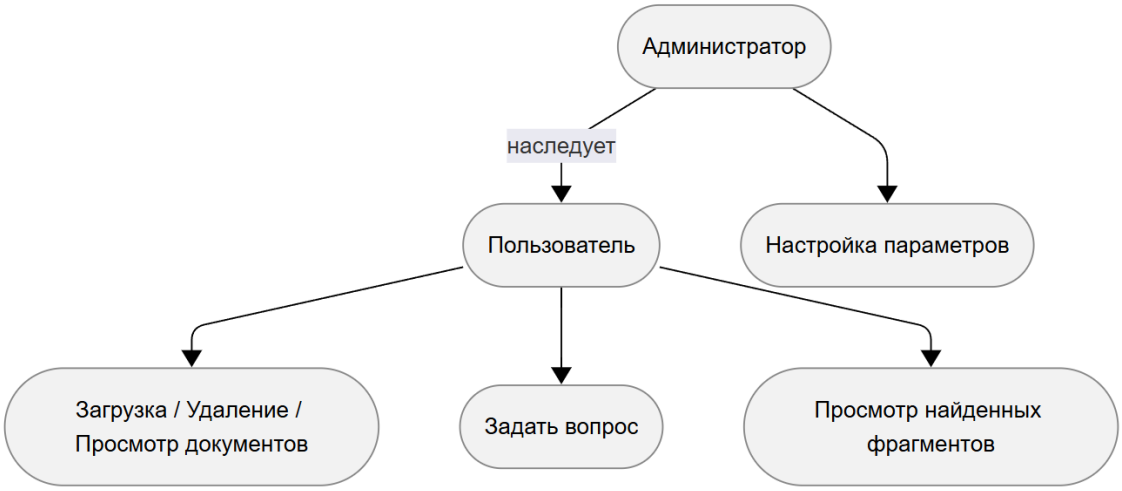


Рисунок 8 – Диаграмма вариантов использования

Следующим шагом была разработана диаграмма последовательностей. Она иллюстрирует, как объекты системы взаимодействуют друг с другом для выполнения разных сценариев работы. На основе диаграммы можно проследить, как данные перемещаются через систему и как компоненты обмениваются сообщениями (см. Приложение Б). Этот этап помог уточнить логику взаимодействий между элементами сервиса.

В дополнение к созданным диаграммам также была составлена диаграмма активностей. Она представляет анализ активностей пользователей, который помог понять, как именно пользователи взаимодействуют с системой в рамках различных сценариев. (см. Рисунок 10)

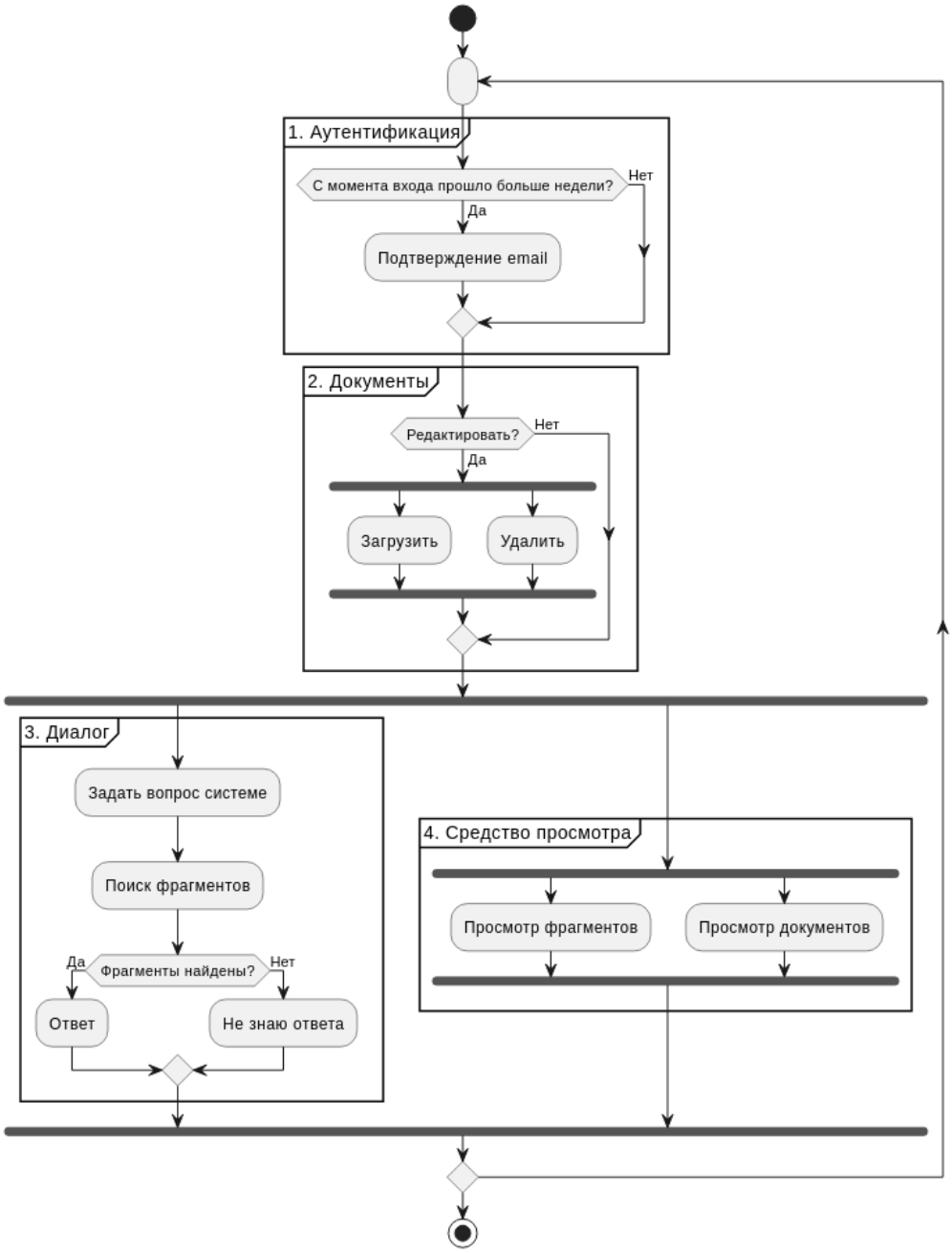


Рисунок 10 – Диаграмма активностей

6 РЕАЛИЗАЦИЯ ПРОТОТИПА

После завершения проектирования был реализован прототип системы. В данной главе приводится описание реализованной системы, охватывающее как серверную, так и клиентскую часть, структуру хранения данных, а также RAG-модуль, играющий ключевую роль в системе.

Прототип реализован с учетом требований и проектных ограничений, описанных в предыдущей главе. В процессе реализации были учтены актуальные практики, ориентированные на промышленные стандарты разработки. Проект прошел несколько итераций разработки и тестирования, в процессе которых некоторые проектные решения были скорректированы. Итоговая архитектура и используемые технологии незначительно отличаются от тех, что были описаны в главе про проектирование системы. В результате были приняты следующие архитектурные и программные решения:

1 Retrieval этап реализован на основе векторного поиска вместо двухступенчатого с применением BM25 [5].

2 Использование СУБД PostgreSQL вместо SQLite. Реализация полнотекстового поиска с помощью SQLite оказалась нетривиальной задачей, поэтому СУБД PostgreSQL была выбрана, как решение, которое лучше подходит под требования проекта и предоставляет необходимый набор инструментов, полноценную документацию и активную поддержку сообществом разработчиков.

3 Клиентская часть разработана с нуля, так как адаптация готового решения вроде ChatUI [21] под требования проекта оказалась неоправданно трудозатратной. При этом в качестве основы для стилистического оформления были частично использованы элементы ChatUI.

4 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. Решение было найдено в использовании библиотеки testcontainers [24]. Она позволяет создавать каждый раз контейнеры с новой базой данных “на лету”

прямо в коде проекта.

6.1 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 однонаправленного протокола передачи данных Server-Sent Events (SSE) [25]. Ключевым элементом архитектуры является RAG-модуль, обеспечивающий возможность семантического поиска и генерации ответов на естественном языке на основе загруженных документов.

6.2 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 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.

6.3 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 описания архитектуры проекта использованы второй и третий уровни: “Контейнеры” и “Компоненты”. Они позволили описать систему в достаточной детализации без избыточных подробностей о кодовой базе проекта. На Рисунке 11 представлена диаграмма контейнеров, показывающая общее взаимодействие клиента, сервера, базы данных и RAG-пайплайна.

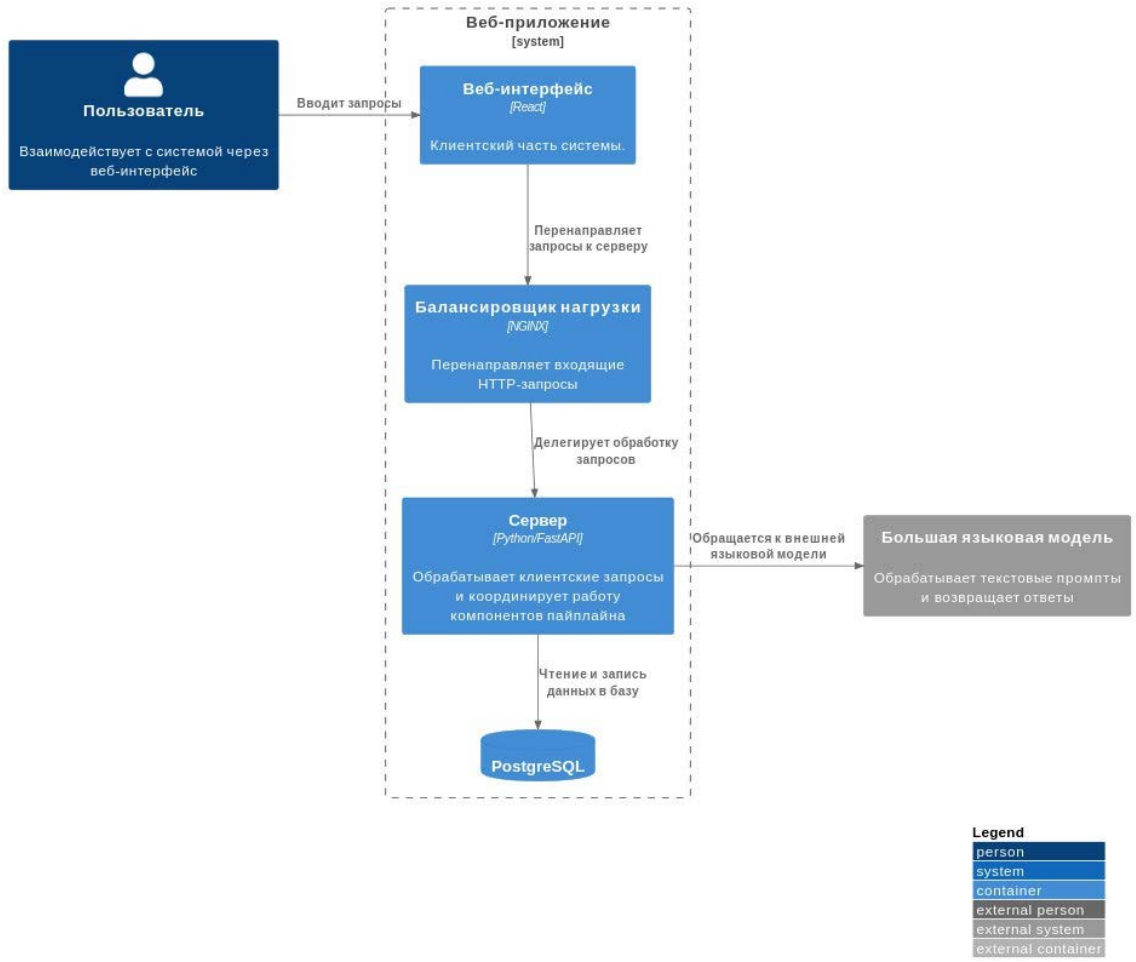


Рисунок 11 – диаграмма C4 (уровень контейнеров)

Следующий уровень детализации – компоненты серверной части (Рисунок 12). На нем можно выделить 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.

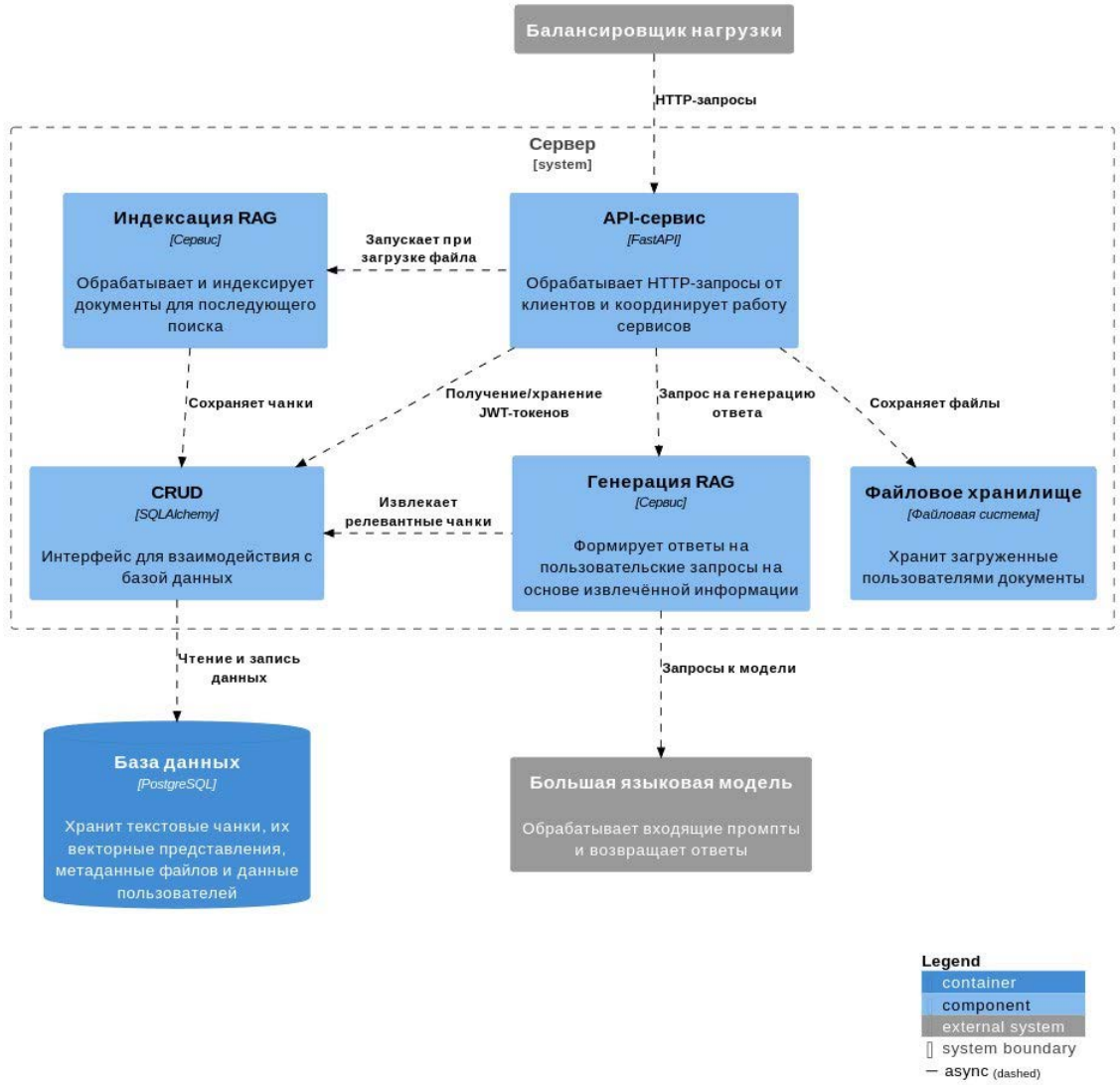


Рисунок 12 – диаграмма C4 (уровень компонентов, сервер)

В дополнение, была разработана более детализированная схема взаимодействия внутри RAG-пайплайна с этапами индексации и генерации. (см. Рисунок 13).

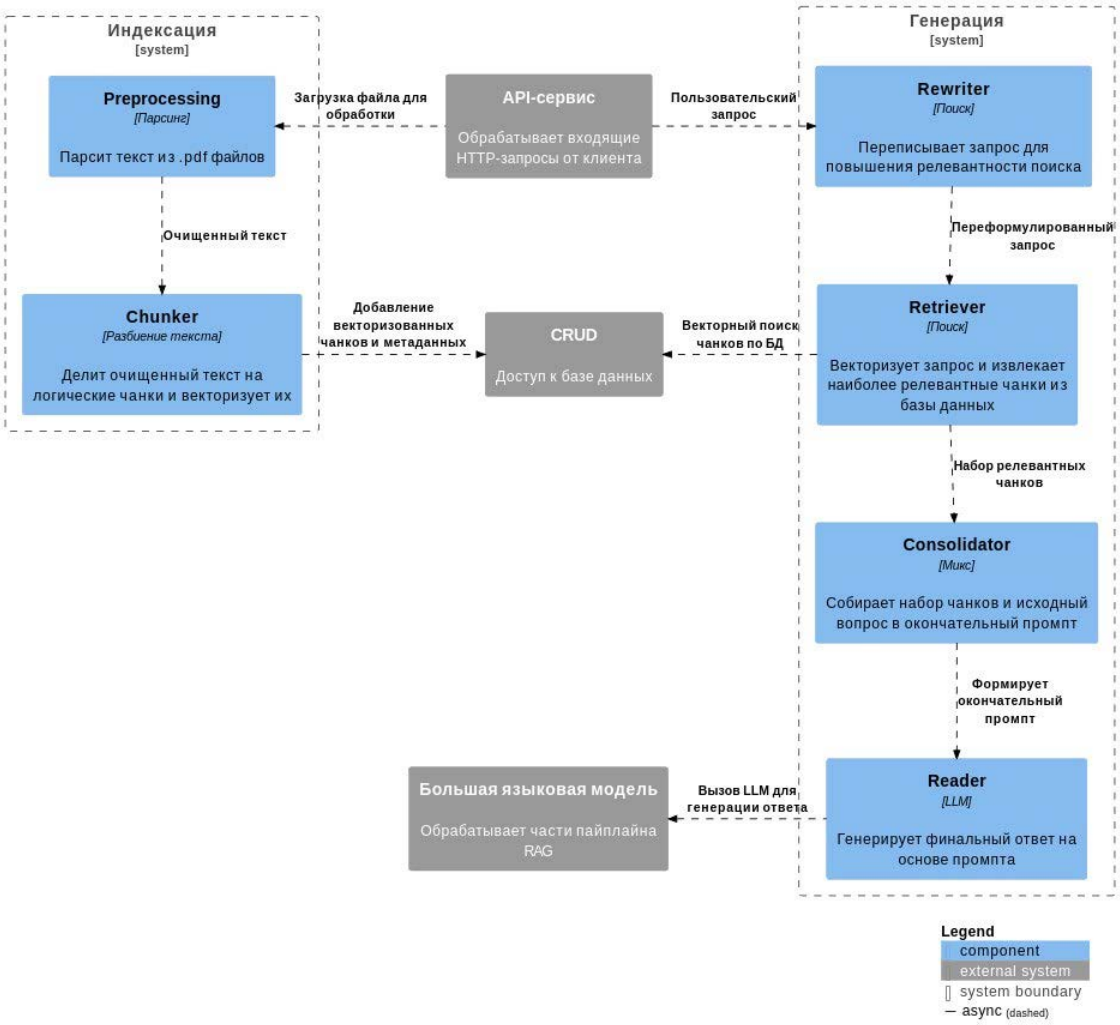


Рисунок 13 - диаграмма C4 (уровень компонентов, RAG-пайплайн)

6.4 Клиентская часть

Клиент (см. Рисунок 14) 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.

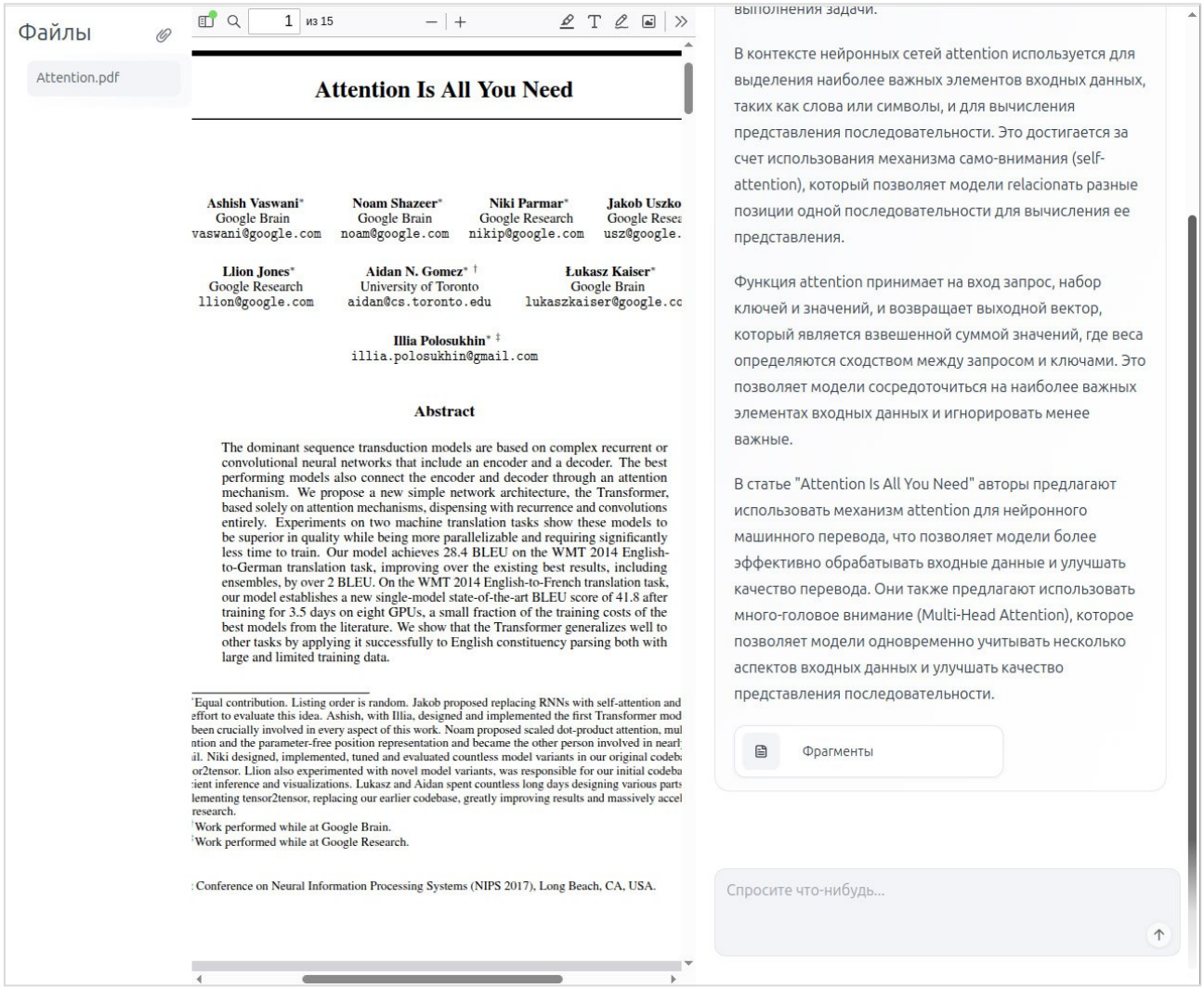


Рисунок 14 – Пользовательский интерфейс веб-приложения

6.5 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.

6.6 RAG-модуль

Работа модуля включает два этапа: Индексация и Генерация (см. Рисунок 13). Первоначально рассматривался классический подход с использованием двухступенчатого поиска на основе BM25. Но несмотря на его простоту и

эффективность в ряде задач, он имеет ограничение при работе с мультиязычными данными. Поскольку как запросы, так фрагменты в базе данных могут быть представлены как на русском, так и на английском языках, алгоритм поиска на основе BM25 не подходит. В качестве альтернативы был реализован семантический поиск с использованием векторных представлений как для запросов, так и для чанков из базы данных. Поиск основан на Bi-Encoder подходе, он позволяет системе находить релевантные фрагменты даже с учетом различия в формулировках и языках.

7 ТЕСТИРОВАНИЕ

Для проверки корректности и надёжности функционала уже в процессе разработки составлялись модульные и интеграционные тесты с использованием библиотеки Pytest [26] и асинхронного клиента из библиотеки HTTPX [27]. 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 Этапы индексации и генерации RAG-модуля.

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

Кроме того, разработан модуль, который проводит бенчмарк системы с помощью LLM-as-a-Judge [18] подхода. Он подключается к модулю RAG и выполняет его тестирование на основе датасета.

7.1 Датасет

Первоначально рассматривался датасет QASPER [17], однако в процессе разработки были выявлены его ограничения: вопросы в датасете слишком общие, к тому же каждый вопрос связан только с одной статьей. Эти ограничения делают его малопригодным для тестирования сложных Retrieval-механизмов. В итоге был выбран датасет FRAMES (Factuality, Retrieval, And reasoning MEasurement Set) [28]. Основные характеристики датасета:

1 824 сложных вопроса, 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 RAG-систем. В статье FRAMES [28] 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 [5], показывают лучшие результаты [28]. Это подтверждает, что добавление поискового механизма может существенно улучшить качество ответов.

Для бенчмарка из датасеты случайным образом были отобраны 50 вопросов, и проиндексированы связанные с ними статьи. Чтобы приблизить эксперимент к реальным условиям, в индекс также добавлены 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.

7.2 Конфигурации

Для сравнительного тестирования были подготовлены три конфигурации вопрос-ответных систем:

1 Модуль RAG из разработанной системы.

2 Версия Agentic RAG. Для его реализации была использована библиотека smolagents [16], а в качестве инструмента подключен поисковой механизм разработанного RAG-модуля. В результате получилась система, способная пошагово рассуждать и извлекать данные, если нужно. Цепочка рассуждений ограничена 6 шагами.

3 LLM без поиска (так как данная версия не использует информационный поиск, метрика Response Relevancy не может быть рассчитана).

Во всех конфигурациях использовалась большая языковая модель openai/gpt-4o-mini [29] и эмбеддинг-модель deepvk/USER-bge-m3 [30].

7.3 Анализ результатов

Стоит отметить, что большие языковые модели склонны к более лояльной оценке своих текстов [31], поэтому для метрик LLM-as-Judge использована

другая модель: google/gemini-pro [32] и библиотека RAGAS [15].

7.3.1 Сравнение метрик на разных конфигурациях

Таблица 1 – Сравнение метрик

	Agentic-RAG	RAG	LLM
Faithfulness	**0.68**	0.54	0.25
Response Relevancy	**0.74**	0.56	-
Answer Correctness	**0.62**	0.48	0.44
Среднее время работы (с)	29	6.5	2.37

7.3.2 Выводы по метрикам

Agentic-RAG показывает лучшие результаты по извлечению фрагментов благодаря множественным вызовам поиска по базе знаний. В то же время агентной системе требуется больше всего времени для ответа. Также это требует больших затрат на токены большой языковой модели. Эти особенности делают агентные системы подобного рода почти неприменимыми в приложениях, которые общаются с пользователем в реальном времени.

LLM без информационного поиска показывает себя ожидаемо гораздо хуже конфигураций, которые имели доступ к внешней базе знаний. Несмотря на то, что метрика Answer Correctness незначительно отличется от версии RAG, метрика Faithfulness значительно отстает, из чего можно сделать вывод, что LLM хоть и дает фактологически верный ответ в ряде случаев, но не может предоставить важных подробностей.

RAG показывает себя средне, являясь скорее компромиссом между чистой LLM и рассуждающими системами.

7.4 Пилотное тестирование

В рамках разработки проекта было проведено короткое пилотное тестирование. Для этого приложение было развернуто глобально в сети Интернет. Доступ к приложению получили 6 студентов НГУ. 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 приложение такого вида в учебном процессе. Анкеты включили в себя 9 вопросов: 6 из них принимали оценку от 0 до 10 и касались разных аспектов приложения, а также общую оценку. Два последних вопроса позволяли пользователю поделиться впечатлениями (см. Приложение В).

Поскольку опрос из 6 человек представляет из себя малую выборку, было решено рассчитать доверительные интервалы для оценок пользователей. Доверительный интервал используется для оценки надежности среднего значения по малой выборке и позволяет узнать диапазон, в котором с большой вероятностью лежит истинное значение для всей выборки. В результате проведенных вычислений (см. Приложение Г) получились результаты с оценкой по разным аспектам приложения (см. Таблица 2):

Таблица 2 – доверительные интервалы (95%) для каждого аспекта

	Среднее	Нижняя оценка	Верхняя оценка
Точность ответов	4.16	2.93	5.39
Скорость работы	5.83	5.04	6.62
Удобство	4.5	3.39	5.6
Полезность	4.0	2.51	5.48
Общая оценка	4.83	3.6	6.06

В результате опроса, можно сделать следующие выводы:

1 Система корректно обрабатывает запросы, однако в некоторых ситуациях дает слишком общие ответы. Особенно это заметно в ситуациях, когда пользователь в своем запросе хочет указать на конкретный структурный элемент текста, например на определенную главу.

2 Пользователи высоко оценили концепт дизайна интерфейса. Особо отметили возможность просмотра PDF-файлов прямо в приложении, а также скорость ответов системы.

ЗАКЛЮЧЕНИЕ

Современная образовательная сфера переживает трансформацию, вызванную 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” [33] 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 подобранным 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 была спроектирована и реализована гибкая масштабируемая система на основе 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, 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 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 Stephenson N. The diamond age: Or, a young lady's illustrated primer. – Spectra, 2003.

3 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.

4 Wang X. et al. Searching for best practices in retrieval-augmented generation //arXiv preprint arXiv:2407.01219. – 2024.

5 Robertson S. E. et al. Okapi at TREC-3 //Nist Special Publication Sp. – 1995. – Т. 109. – С. 109.

6 5 levels of text splitting // GitHub URL: https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb (дата обращения: 10.05.2025).

7 Zhao J. et al. Meta-chunking: Learning efficient text segmentation via logical perception //arXiv preprint arXiv:2410.12788. – 2024.

8 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.

9 Gao L. et al. Precise zero-shot dense retrieval without relevance labels //Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers). – 2023. – С. 1762-1777.

10 Glass M. et al. Re2G: Retrieve, rerank, generate //arXiv preprint arXiv:2207.06300. – 2022.

11 Reimers N., Gurevych I. Sentence-bert: Sentence embeddings using siamese bert-networks //arXiv preprint arXiv:1908.10084. – 2019.

12 Rosa G. et al. In defense of cross-encoders for zero-shot retrieval //arXiv

preprint arXiv:2212.06121. – 2022.

13 Zhang R. et al. Toward agentic AI: generative information retrieval inspired intelligent communications and networking //arXiv preprint arXiv:2502.16866. – 2025.

14 Yao S. et al. React: Synergizing reasoning and acting in language models //International Conference on Learning Representations (ICLR). – 2023.

15 Yu H. et al. Evaluation of retrieval-augmented generation: A survey //CCF Conference on Big Data. – Singapore : Springer Nature Singapore, 2024. – С. 102-120.

16 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.

17 Smolagents // GitHub URL: https://github.com/huggingface/smolagents (дата обращения: 10.05.2025).

18 Dasigi P. et al. A dataset of information-seeking questions and answers anchored in research papers //arXiv preprint arXiv:2105.03011. – 2021.

19 Gu J. et al. A survey on llm-as-a-judge //arXiv preprint arXiv:2411.15594. – 2024.

20 ML System Design Doc // GitHub URL: https://github.com/IrinaGoloshchapova/ml_system_design_doc_ru/blob/main/ML_System_Design_Doc_Template.md (дата обращения: 10.05.2025).

21 ChatPDF // ChatPDF URL: https://www.chatpdf.com/ru (дата обращения: 10.05.2025).

22 ChatUI // GitHub URL: https://github.com/huggingface/chat-ui (дата обращения: 10.05.2025).

23 SQLite // Wikipedia URL: https://en.wikipedia.org/wiki/SQLite (дата обращения: 10.05.2025).

24 SQLAlchemy // GitHub URL: https://github.com/sqlalchemy/sqlalchemy (дата обращения: 10.05.2025).

25 Testcontainers Python // GitHub URL:

https://github.com/testcontainers/testcontainers-python (дата обращения: 10.05.2025).

26 Server-sent events // Wikipedia URL: https://ru.wikipedia.org/wiki/Server-sent_events (дата обращения: 10.05.2025).

27 Pytest // Wikipedia URL: https://en.wikipedia.org/wiki/Pytest (дата обращения: 10.05.2025).

28 HTTPX // GitHub URL: https://github.com/projectdiscovery/httpx (дата обращения: 10.05.2025).

29 Krishna S. et al. Fact, fetch, and reason: A unified evaluation of retrieval-augmented generation //arXiv preprint arXiv:2409.12941. – 2024.

30 GPT-4o mini // OpenAI URL: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ (дата обращения: 10.05.2025).

31 deepvk/USER-bge-m3 // HuggingFace URL: https://huggingface.co/deepvk/USER-bge-m3 (дата обращения: 10.05.2025).

32 Xu W. et al. Pride and prejudice: LLM amplifies self-bias in self-refinement //arXiv preprint arXiv:2402.11436. – 2024.

33 Gemini 2.5 Pro // Google DeepMind URL: https://deepmind.google/technologies/gemini/pro/ (дата обращения: 10.05.2025).

34 Искусственный интеллект и высшее образование: возможности, практики и будущее // Яндекс Образование URL: https://education.yandex.ru/aihighreport (дата обращения: 10.05.2025).

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

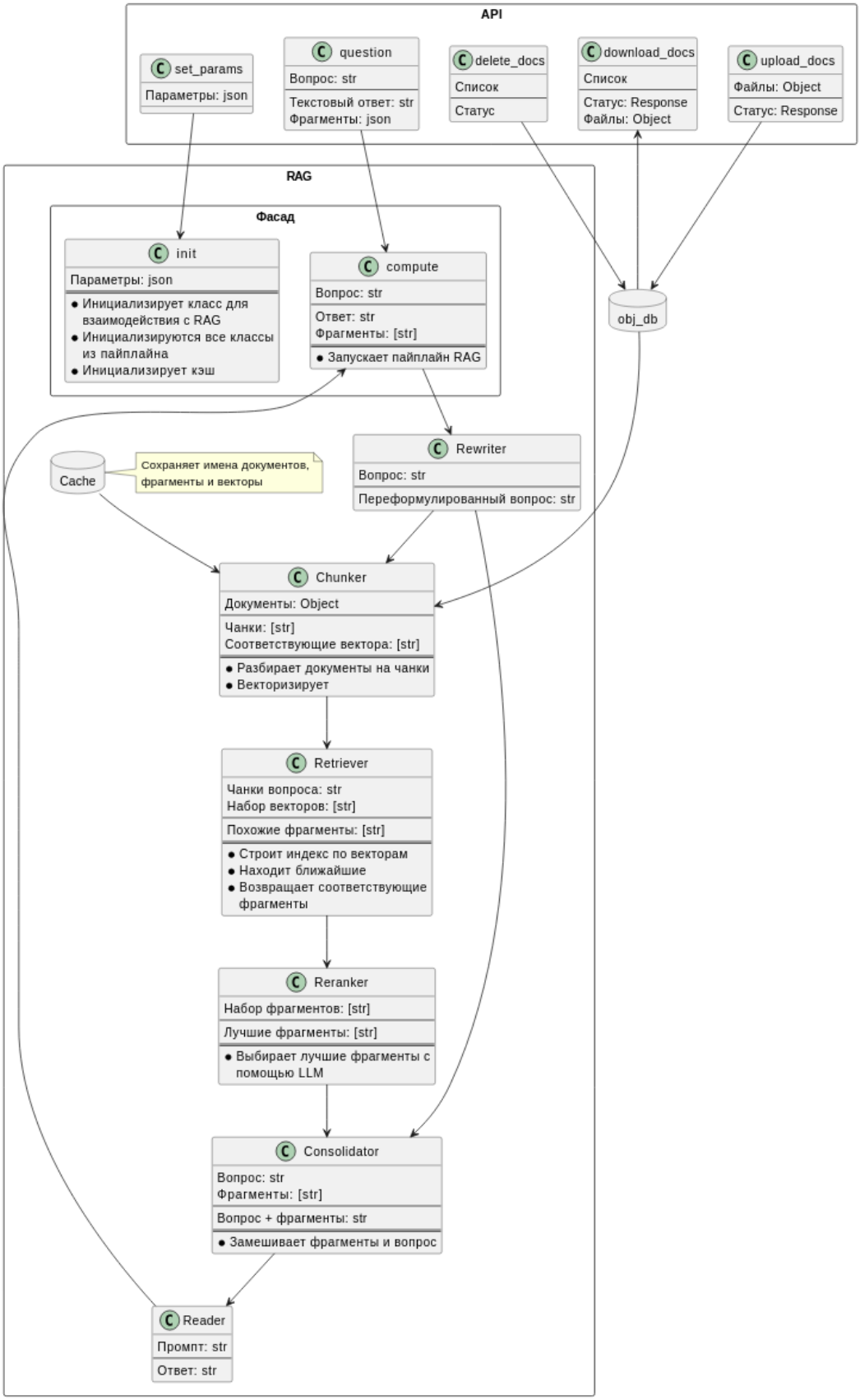
Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

_____ _____

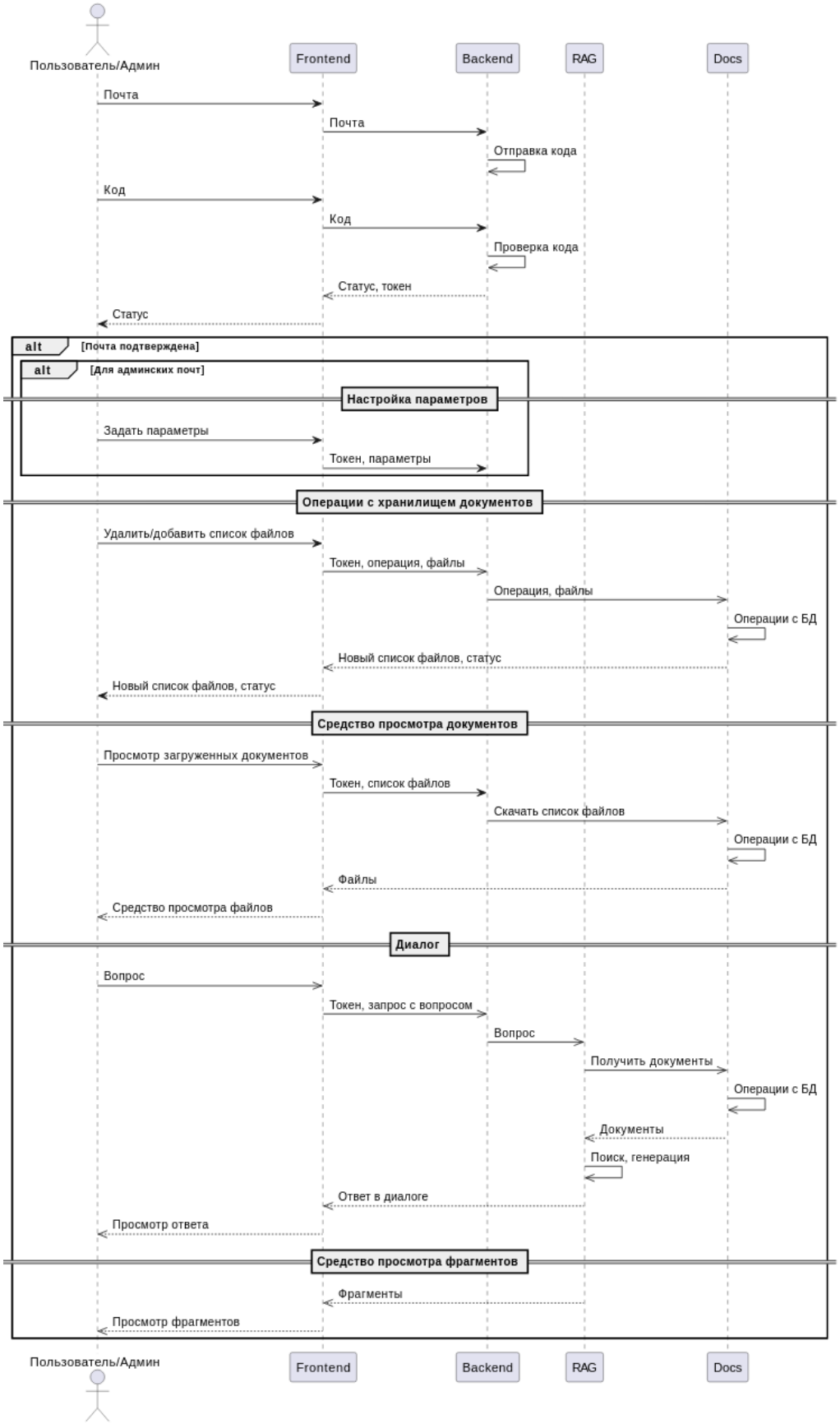
ФИО студента *Подпись студента*

« ____ »_____20 __г.
(заполняется от руки)

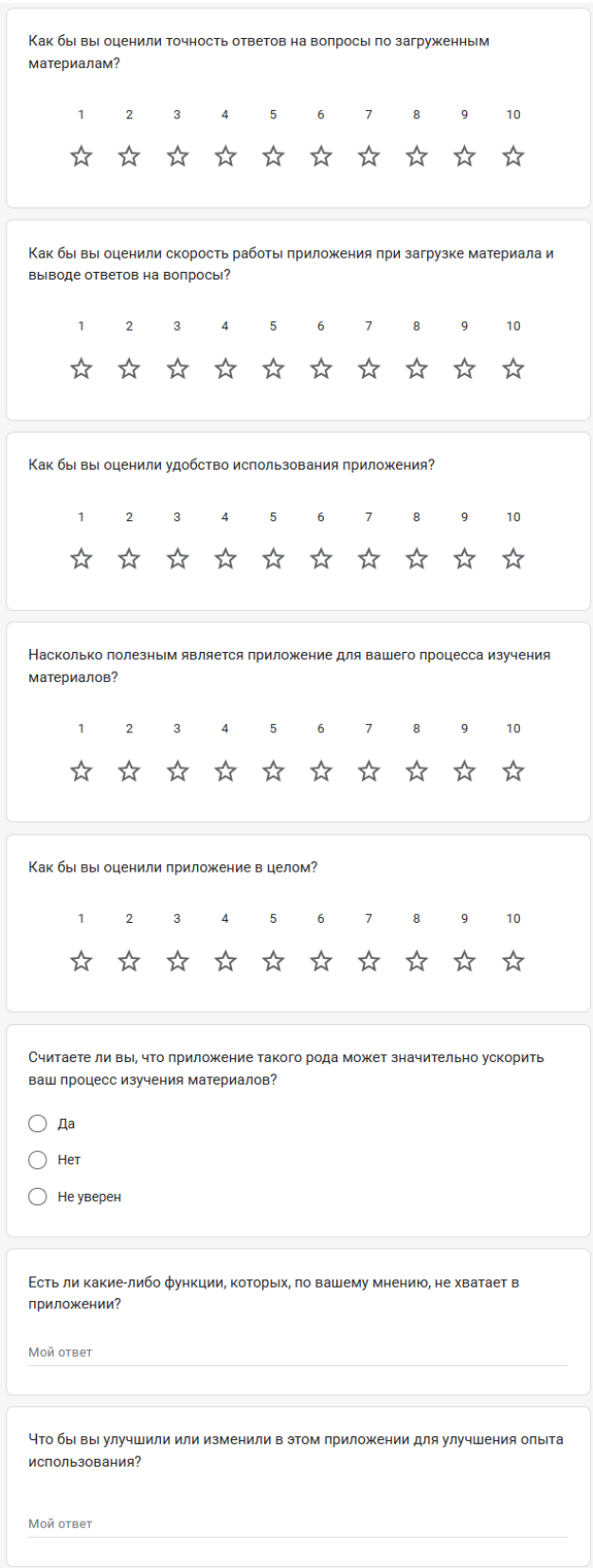
ПРИЛОЖЕНИЕ А
Диаграмма компонентов серверной части



ПРИЛОЖЕНИЕ Б
Диаграмма последовательностей



ПРИЛОЖЕНИЕ В
Анкета для опроса пользователей



ПРИЛОЖЕНИЕ Г

Программный код для подсчета доверительных интервалов

```
import numpy as np
import scipy.stats as stats
import pandas as pd

# Входные данные - результаты опроса
data = np.array([
    [3, 6, 4, 4, 5],
    [4, 5, 3, 5, 6],
    [4, 6, 5, 3, 3],
    [6, 7, 5, 4, 6],
    [3, 5, 6, 2, 4],
    [5, 6, 4, 6, 5],
])

# Параметры
confidence = 0.95  # 95% доверительный интервал
n = data.shape[0]

means = data.mean(axis=0)
std_err = stats.sem(data, axis=0)
h = std_err * stats.t.ppf((1 + confidence) / 2.0, n - 1)

intervals = pd.DataFrame({
    'Mean': means,
    'CI Lower': means - h,
    'CI Upper': means + h
})
print(intervals)
```