Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

К ЗАЩИТЕ ДОПУСТИТЬ
Зав. кафедрой АСУ
канд. техн. наук, доцент
В.В. Романенко

(подпись)
«____» _____ 2025 г.
(дата)

ПРОГРАММНЫЙ КОМПЛЕКС ИНТЕЛЛЕКТУАЛЬНОГО ПОИСКА В КОРПОРАТИВНЫХ ДОКУМЕНТАХ

Бакалаврская работа
По направлению подготовки 09.03.01
«Информатика и вычислительная техника»

Выполнил: студент гр. 431-3
_____ Е.П. Бекиш
(подпись) (И.О. Фамилия)
«24» июня 2025 г.
(дата)
Руководитель:
доцент каф. АСУ ТУСУР, к.т.н.
(должность, ученая степень, звание)
_____ А.Я. Суханов
(подпись) (И.О. Фамилия)
«26» июня 2025 г.
(дата)

Томск 2025

Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования

ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра автоматизированных систем управления (АСУ)

УТВЕРЖДАЮ
Зав. кафедрой АСУ
канд. техн. наук, доцент
В.В. Романенко

(подпись)
«____» _____ 2025 г.
(дата)

ЗАДАНИЕ

на бакалаврскую работу

студенту гр. 431-3 факультета систем управления

Бекишу Егору Павловичу
(Ф.И.О. студента)

1. Тема бакалаврской работы (БР): Программный комплекс интеллектуального поиска в корпоративных документах
(утверждена приказом по вузу от « 26 » июня 2025г. № 3395 ст).
2. Срок сдачи студентом законченной БР: « 28 » июня 2025г.
3. Исходные данные к работе:
3.1. ОС ТУСУР 01-2021.
3.2. Документация к фреймворку «Streamlit».
3.3. Документация «GigaChat».
3.4. Документы «Юридические документы GigaChat».
4. Содержание расчетно-пояснительной записки/перечень подлежащих разработке вопросов:
4.1. Анализ требований.
4.2. Определение спецификаций.

4.3. Проектирование.

4.4. Кодирование.

4.5. Тестирование.

5. Перечень графического материала (с точным указанием обязательных листов презентации):

5.1. Постановка цели и задачи.

5.2. Обзор предметной области.

5.3. Функциональные требования.

5.4. Обзор аналогов.

5.5. Стек технологий.

5.6. Проектирование.

5.7. Реализация.

5.8. Демонстрация.

5.9. Тестирование.

6. Дата выдачи задания: «10» февраля 2025г.

Руководитель бакалаврской работы

доцент каф. АСУ ТУСУР, к.т.н.		А.Я. Суханов
(должность, ученая степень, звание)	(подпись)	(И.О. Фамилия)

Задание принял к исполнению: «10» февраля 2025г.

Студент гр. 431-3		Бекиш Е.П.
	(подпись)	(Ф.И.О.)

Реферат

Бакалаврская работа содержит 77 страниц, 50 рисунков, 0 таблиц, 30 источников, 7 приложений.

ПРОГРАММНЫЙ КОМПЛЕКС ДЛЯ ПОИСКА РЕЛЕВАНТНОЙ ИНФОРМАЦИИ, ВЗАИМОДЕЙСТВИЕ И ОБРАБОТКА ДОКУМЕНТОВ

Объект разработки: представляет собой использование языковых моделей.

Предмет разработки: методы и технологии внедрения языковых моделей в корпоративные документы, представленных в виде списка документов формата txt, pdf, docx, doc, pptx, ppt.

Цель работы: разработать программный комплекс, который сокращает время доступа сотрудников к корпоративной информации за счет обработки запросов к внутренним документам предприятия на естественном языке и их обработка средствами ИИ.

Полученный результат работы: внедрение системы в корпорации различного рода деятельности для повышения эффективности работоспособности сотрудников.

Область применения разработки: система предназначена для корпораций с объемным электронным документооборотом.

Бакалаврская работа выполнена в текстовом редакторе – Microsoft Word и представлена в электронной версии в электронной образовательной среде ТУСУРа.

Abstract

The bachelor's thesis contains 77 pages, 50 figures, 0 table, 30 sources, and 7 appendices.

SOFTWARE PACKAGE FOR SEARCHING RELEVANT INFORMATION, INTERACTION AND DOCUMENT PROCESSING

Object of development: represents the use of language models.

Subject of development: methods and technologies for implementing language models in corporate documents presented as a list of documents in txt, pdf, docx, doc, pptx, ppt format.

The purpose of the work: to develop a software package that reduces employee access time to corporate information by processing requests to internal enterprise documents in natural language and processing them using AI.

The result of the work: the implementation of the system in corporations of various types of activities to improve the efficiency of employees.

Scope of development: the system is designed for corporations with extensive electronic document management.

The bachelor's thesis was completed in a Microsoft Word text editor and presented in an electronic version in the electronic educational environment of TUSUR.

Оглавление

Введение

Языковые модели [1] являются важной и активно развивающейся областью, которая за последние несколько лет привлекает большое внимание промышленников, научных и академических кругов.

В современном мире объемы информации с каждым днем растут все больше и больше, поэтому анализировать информацию становится труднее, как и выбрать более корректную. В организациях существуют собственные корпоративные документы, которые имеют тоже не маленький объем. Поэтому, чтобы повысить собственное качество и работоспособность им бы хотелось избегать анализа лишнего материала, т.к. на нее тратится большое время.

Объект разработки представляет собой языковые модели. Предметом разработки является использование методов и технологий языковых моделей в корпоративных документах.

В связи с этим, в качестве основной задачи выступает создание системы, которая позволяет сотруднику искать ту информацию, которая релевантная. Осуществляться будет за счет подключения и внедрения разработанной системы, которая будет извлекать необходимые данные их корпоративных документов.

Целью работы является сокращение времени доступа сотрудников к корпоративной информации за счет обработки запросов к внутренним документам предприятия на естественном языке и средствами ИИ.

Для достижения поставленной цели необходимо выполнить следующие задачи:

1. Анализ необходимых требований.
2. Реализация функциональных возможностей и выработка спецификаций.
3. Проектирование.
4. Реализация системы.

5. Тестирование.

Результатом выполненной работы является программный комплекс, который находит нужную информацию на поставленный запрос. Программный комплекс представляет собой чат, в который сотрудник может задавать вопрос, добавлять собственные документы, просматривать внутренние документы и искать информацию в ней.

1 АНАЛИЗ ТРЕБОВАНИЙ

1.1 Проблематика

За последние несколько десятилетий произошел значительный рост объемов информации, что потребовало тщательного анализа поступающих данных, существенно замедляя как образовательный, так и рабочий процесс. В этих условиях традиционные методы работы с информацией становятся неэффективными и требуют значительных ресурсов для поддержания актуальности и доступности информации. Языковые модели (LLM), такие как ChatGPT [2], предлагают инновационные решения для автоматизации обработки и анализа данных, улучшения поиска и доступа к знаниям для пользователей и сотрудников компаний, а также поддержки принятия решений. Однако, несмотря на очевидные преимущества, многие пользователи и компании еще не полностью осознали потенциал LLM и сталкиваются с вызовами при их внедрении.

1.2 Развитие больших языковых моделей

1.2.1 Появление обработки естественного языка

В 1950-м году Алан Тьюринг опубликовал статью «Computing Machinery and Intelligence» [3], в которой рассматривал вопрос: «Могут ли машины мыслить?», где, для начала, нужно определить термины в данном вопросе – «мыслить» и «машина». В дальнейшем выдвинул идею, что человек может взаимодействовать с машиной на естественном языке. Спустя 16 лет была опубликована первая программа по генерации текста ELIZA, которая имитировала психотерапевта, используя шаблоны и правила.

В 70 – 80-х годах развивался метод машинного обучения на базе правил, где данные правила задавались вручную, но системы на тот момент не поддерживали масштабируемость.

1.2.2 Развитие векторного представления слов

Векторное представление слов – вектор в пространстве с фиксированной размерностью. На вход подаются коллекция документов, на выходе получаем векторное представление из коллекции документов.

One-hot encoding [4] – метод представления в векторном виде. Создается словарь фиксированного размера N, где для каждого слова из словаря соответствует вектор размера N, каждый вектор состоит из одной единицы на i-м месте и $(N-1)$ нулей, где i – номер слова в словаре. Таким образом, запись данного метода представлена на рисунке 1.1.

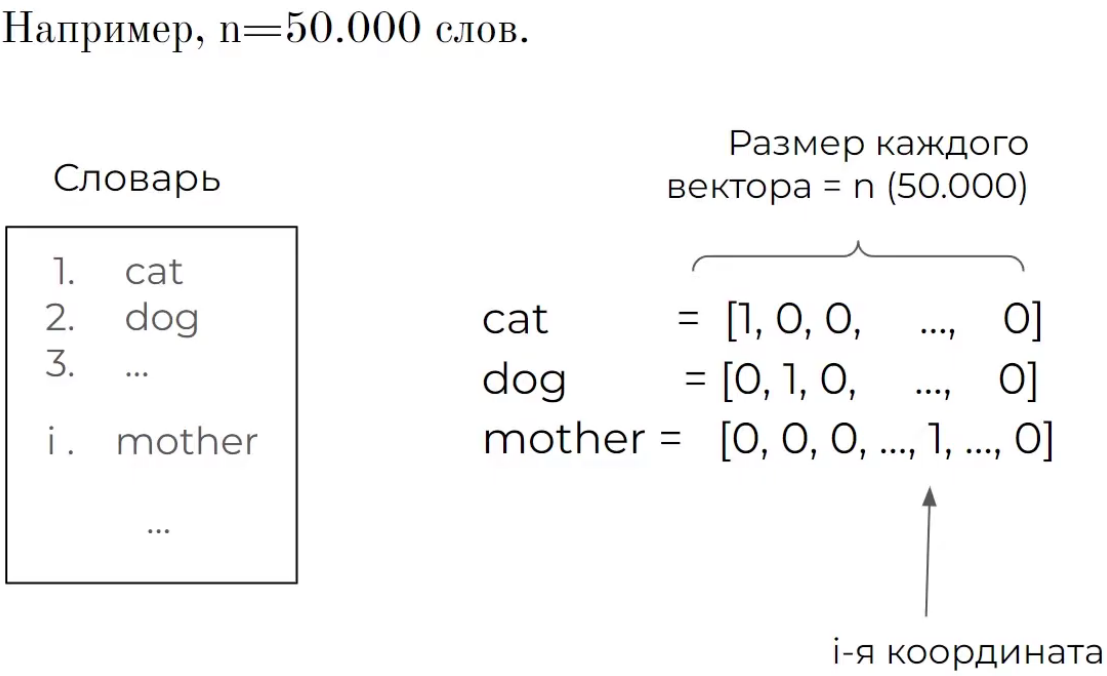

Рисунок 1.1 – One-hot encoding

Перейдем к методу, который учитывает данную важность. TF-IDF [5] – вычисляет важность каждого слова в документе относительно количества его употреблений в данном документе и во всей коллекции текстов. Этот метод позволяет выделить ключевые слова и понять, какие слова имеют больший вес для определенного документа в контексте всей коллекции. Или же более простыми словами – мера важности слова t для документа d среди документов D.

Алгебраическая форма записи выгляди следующим образом:

$$tf - idf(t, d, D) = tf(t, d) \times idf(t, D) \quad (1.1)$$

где t – входное слово;

d – текущий документ;

D – коллекция документов;

$tf(t, d)$ – относительная частота встречаемого слова в документе;

$idf(t, D)$ – обратная частота встречаемого слова в наборе документов.

Относительная частота рассчитывается следующим образом:

$$tf(t, d) = \frac{n_t}{\sum_k n_k} \quad (1.2)$$

где t – входное слово;

d – документ, в котором ищется слово t;

n_t – количество слова t в документе d;

$\sum_k n_k$ – количество слов в документе d.

Обратная частота рассчитывается следующим образом:

$$idf(t, D) = \log \frac{|D|}{|\{d_i \in D | t \in d_i\}|} \quad (1.3)$$

где t – входное слово;

D – коллекция документов, их количество;

$|\{d_i \in D | t \in d_i\}|$ – количество документов, где слово t встречается.

Следовательно, значение обратной частоты показывает, чем оно больше, т.е. количество слов t встречается реже во всех документах, тем оно является более важным по смыслу в текущем документе, в противном случае, как было сказано про артикли и предлоги до этого, их количество велико, что говорит нам о том, что их важность очень мала и не несет никакой смысловой нагрузки. Теперь, когда можно найти важность каждого слова среди документов, строятся векторы документов из данных значений, который имеет размер словаря N и на i-й позиции вектора будет стоять значение i-го слова данного документа. Для наглядности пример можно увидеть на рисунках 1.2 – 1.3.

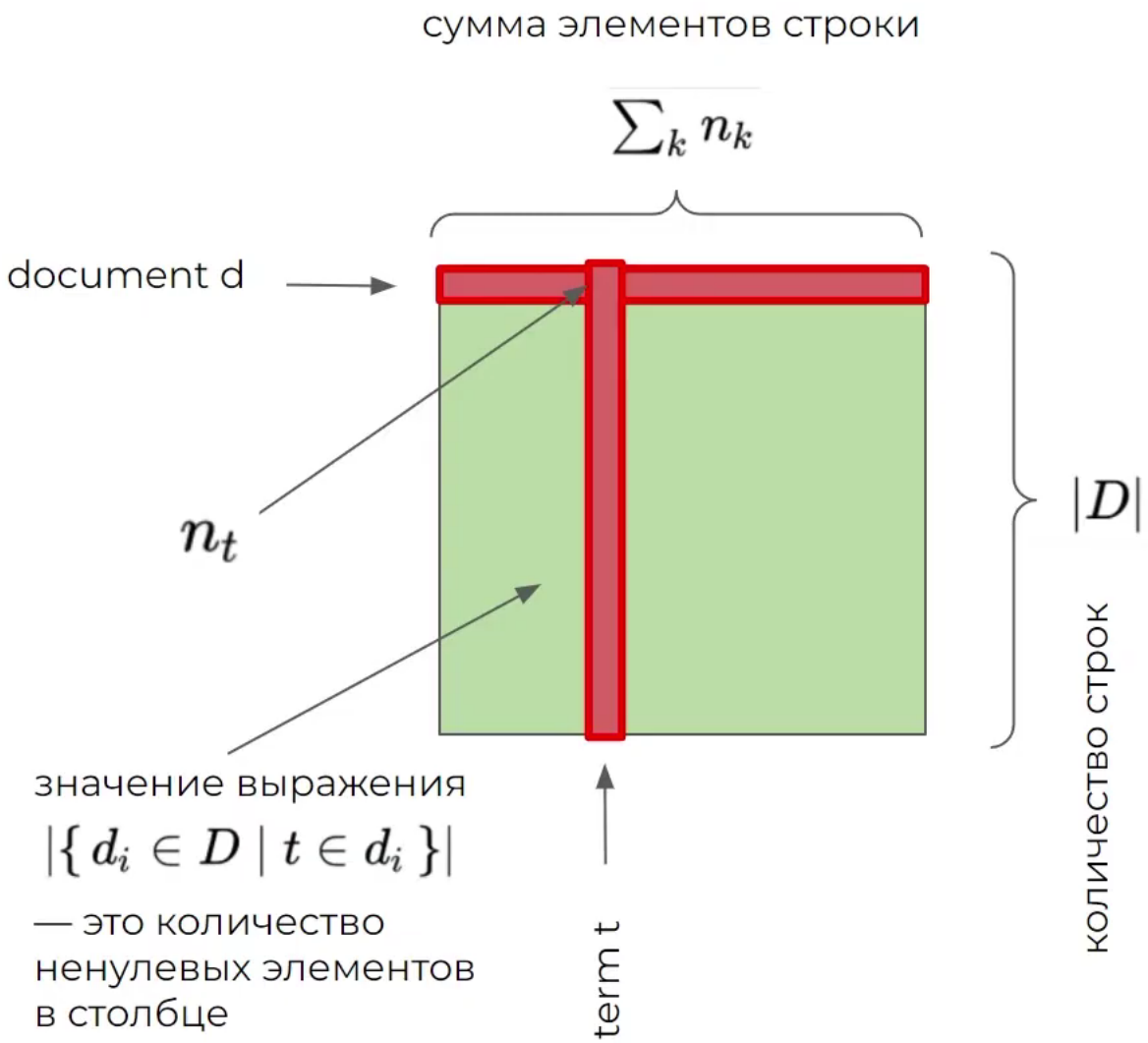

Рисунок 1.2 – Матрица TF-IDF

1. **a dog eats meat**
2. **a dog hunts cat**

TF-IDF векторы для этих предложений:

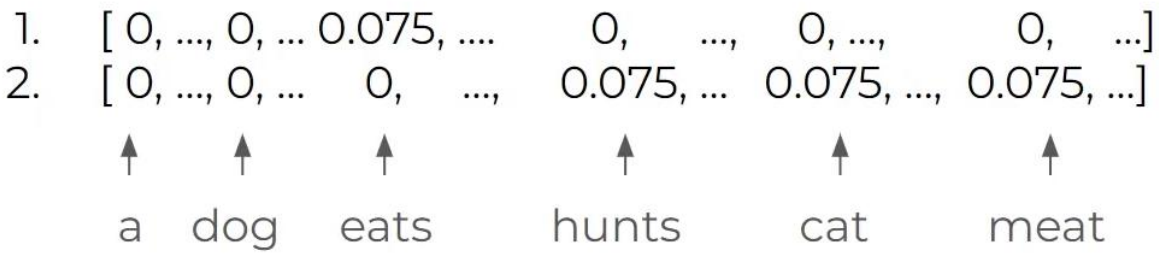

Рисунок 1.3 – Векторное представление TF-IDF

На основе предыдущего метода была разработана модификация TF – IDF – BM25 [6] – полнотекстовый поиск заключается в нахождении *наиболее релевантных* запросу документов из множества вариантов.

Алгебраическая форма записи выглядит следующим образом:

$$score(D, Q) = \sum_{i=1}^{n} IDF(q_i) \cdot TF(q_i, D) \quad (1.4)$$

где D – документ;

Q – запрос;

$IDF(q_i)$ – обратная частота документа;

$TF(q_i, D)$ – частота слова в документе.

Часто слова в документе выглядит следующем образом:

$$TF(q_i, D) = \frac{f(q_i, D) \cdot (k_1 + 1)}{f(q_i, D) + k_1} \cdot \frac{1}{\left(1 - b + b \cdot \frac{|D|}{avgdl}\right)} \quad (1.5)$$

где q_i – запрос;

D – документ;

$f(q_i, D)$ – частота запроса в документе;

k_1, b – свободные коэффициенты ($const$, $k_1 \in [1.2; 2.0]$, $b \in [0; 1]$);

$|D|$ – длина документа;

$avgdl$ – средняя длина документа в нашей коллекции.

Обратная частота документа выгляди следующим образом:

$$IDF(q_i) = ln\left(\frac{N - n(q_i) + 0.5}{n(q_i) + 0.5}\right) \quad (1.6)$$

где N – общее количество элементов в коллекции;

$n(q_i)$ – количество документов, содержащих элемент запроса;

$N - n(q_i)$ – это количество документов, не содержащих элемент запроса.

Сам алгоритм BM25, как и было выше сказано основан на принципе вероятностного ранжирования – если извлечённые документы упорядочены по уменьшению вероятности релевантности данных, то эффективность системы оптимальна для этих данных.

1.2.3 Виды языковых моделей

С основными методами нахождения ранжированных слов, предложений и документов разобрались. Теперь необходимо как-то сформировать верное сформулированное и понятное по смыслу предложение. Одна из моделей – N-граммы – это последовательность из N подряд идущих символов, звуков, слов, слогов и т.п. в тексте [7]. Данная модель может быть:

1. Униграммой (1-граммы) – отдельные символы, слова и т.п.
2. Биграммой (2-граммы) – пара слов.
3. Триграммы (3-граммы) – тройка слов.

Статическая модель, предсказывающая или предугадывающая следующее слово, основанное на вероятностной модели. Рассчитывается по цепному правилу:

$$P(w_1^n) = P(w_1)P(w_2|w_1)\cdots P(w_n|w_1^{n-1}) = \prod_{k=1}^{n} P\left(w_k\middle|w_1^{k-1}\right) \quad (1.7)$$

где w_n – слово в тексте;

$P(w_1^n)$ – общий случай N-граммы.

Как можем заметить, что вероятность слова w_n зависит от предыдущего w_1^{n-1}, а не от всего предыдущего текста, что было до этого, тогда общий случай выглядит следующим образом:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1},w_n)}{count(w_{i-1})} \quad (1.8)$$

Возникает проблема нулевых вероятностей, говорит о том, что если n-грамма не была в обучающихся данных, то вероятность равно нулю и модель не работает. Существует аддитивное сглаживание Лапласа, которое исправляет данную проблему. Алгебраическая запись выглядит следующим образом:

$$P(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)+1}{count(w_{i-1})+V} \quad (1.9)$$

где V – размер словаря.

Как и было сказано, мы можем предугадывать следующее слова, учитывая только предыдущее слово, но нам необходимо учитывать большую

часть контекста. Рекуррентная нейронная сеть запоминает предыдущую информацию и передает выход одного этапа на вход следующего, таким образом учитывая предыдущие слова для выбора наиболее подходящего последующего. Память о предшествующих этапах позволяет модели лучше воспринимать контекст и точнее предсказывать последующие элементы [8].

Базовым элементом вычислений в рекуррентных нейронных сетях служит рекуррентный блок. Внутри него хранится скрытая переменная, содержащая сведения о прошлых элементах последовательности. Эти блоки способны сохранять данные о предыдущих шагах благодаря возврату своего скрытого состояния, что даёт возможность выявлять временные взаимосвязи между элементами. На рисунке 1.4 представлен принцип работы рекуррентного нейрона.

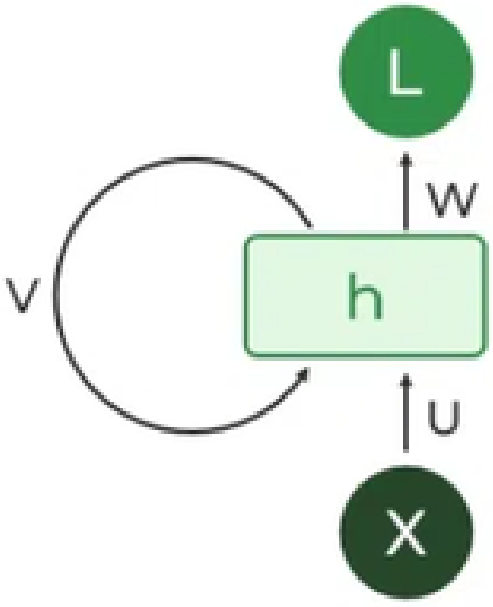

Рисунок 1.4 – Рекуррентный нейрон

Так как рекуррентные нейронные сети работают с последовательными данными, для оптимизации весовых коэффициентов применяется метод обратного распространения ошибок во времени (BPTT). Значение целевой функции потерь $L(\theta)$ определяется последним скрытым состоянием, причём каждое промежуточное скрытое состояние зависит от предыдущего, формируя непрерывную цепь взаимозависимых состояний, как показано на рисунке 1.5.

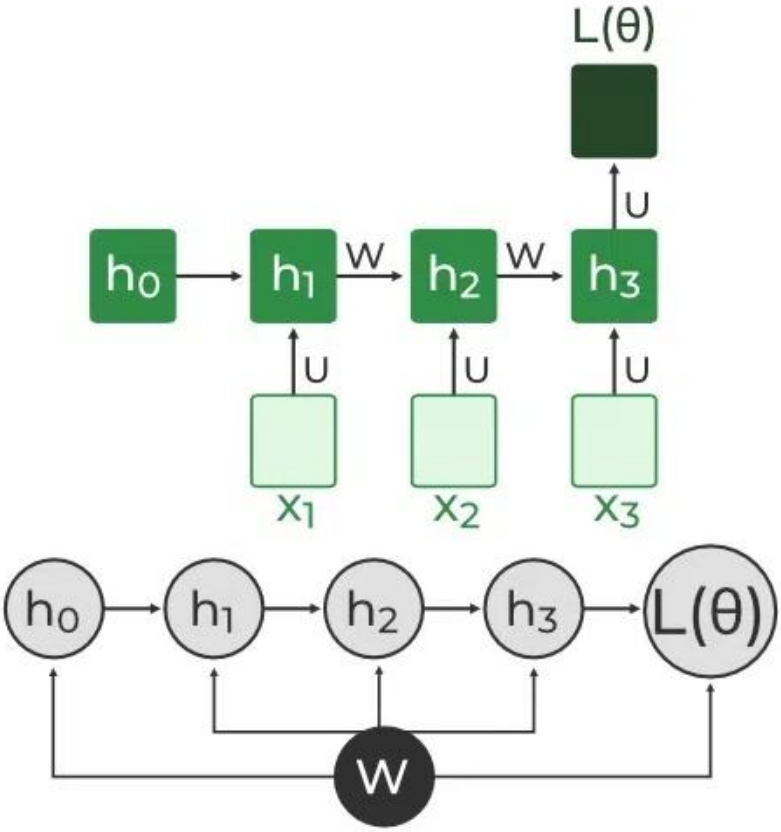

Рисунок 1.5 – Обратное распространение по времени

При применении метода обратного распространения во времени (BPTT), изменения градиента передаются обратно по каждому временному интервалу. Это необходимо для обновления параметров сети с учётом зависимостей, существующих между отдельными моментами времени.

Хотя рекуррентные нейронные сети отлично подходят для анализа последовательных данных, но они испытывают трудности при обучении, сталкиваясь с двумя ключевыми проблемами, т.е. проблемой исчезающего градиента и проблемой взрывающегося градиента.

Данная проблема решается усовершенствованной версией RNN – Long Short-Term Memory (LSTM) [9].

Благодаря механизму забывающих ворот, LSTM способен удалять лишнюю информацию, предотвращая проблему затухания градиента. С помощью этого LSTM успешно справляется с гораздо большими объемами последовательных данных по сравнению с обычными RNN. Архитектура LSTM представлена на рисунке 1.6.

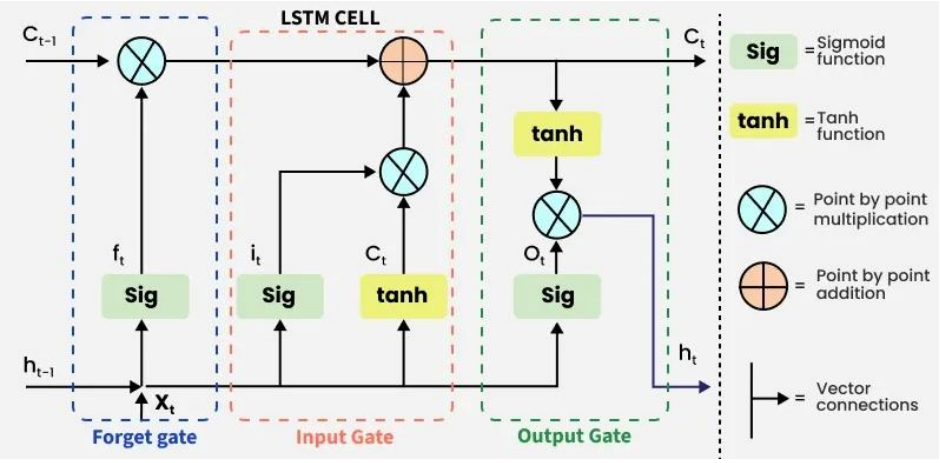

Рисунок 1.6 – Модель LSTM

Входы и выходы управляют тем, какую именно информацию следует хранить в долговременной памяти и каким образом её следует применять. Это способствует более точной и эффективной передаче данных внутри сети.

Рассмотрим каждый этап архитектуры LSTM.

Забыть о воротах. Ненужные данные в состоянии ячейки удаляются через механизм забывания. На вход подаются два сигнала: текущий входной вектор x_t и предыдущее выходное состояние h_{t-1}. Они проходят через весовые матрицы, к ним добавляется смещение, а затем результат преобразуется с помощью функции активации, выдающей бинарное значение. Если выход равен 0, соответствующая информация стирается, а если 1 — сохраняется для дальнейшего использования. Данный механизм представлен на рисунке 1.7.

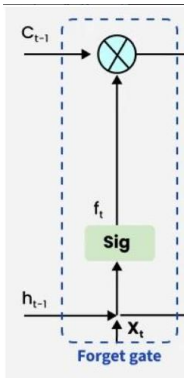

Рисунок 1.7 – Забытые ворота

Входной затвор. Полезная информация добавляется к состоянию ячейки через входной вентиль. Сначала данные обрабатываются сигмоидальной

функцией, которая определяет, какие значения следует сохранить, используя входные сигналы h_{t-1} и x_t. Затем с помощью гиперболического тангенса tanh формируется вектор в диапазоне от -1 до 1, включающий все возможные комбинации h_{t-1} и x_t. На последнем этапе этот вектор умножается на отфильтрованные значения, в результате чего получается обновленная полезная информация. Данный механизм представлен на рисунке 1.8.

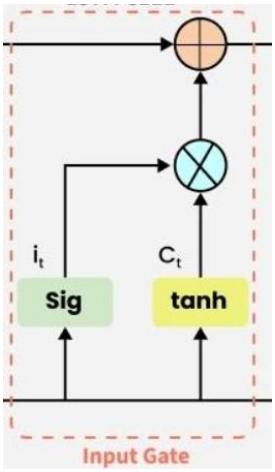

Рисунок 1.8 – Входной затвор

Выходной затвор. Отвечает за выбор полезной информации из текущего состояния ячейки для формирования итоговых данных. Сначала состояние ячейки обрабатывается функцией tanh, создавая вектор значений. Затем сигмоидальная функция определяет, какие элементы этого вектора следует сохранить, анализируя входные сигналы h_{t-1} и x_t. На финальном этапе отфильтрованные значения умножаются на полученный вектор, и результат передается на выход сети, а также поступает в следующую ячейку. Механиз представлен на рисунке 1.9.

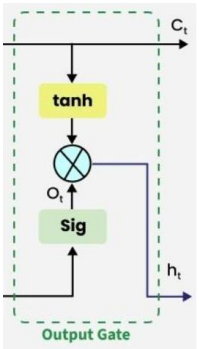

Рисунок 1.9 – Выходной затвор

Как можно было заметить, текущие модели обрабатывают данные последовательно, что значительно уменьшают эффективность, что приводит к тому, что обработка больших последовательностей очень медленная. На смену данных моделей пришли трансформеры [10], которые устраняют данную проблему, добавляя параллелизм и масштабируемость в обработке.

Модель трансформера включает в себя два основных компонента: кодировщик (энкодер) и декодировщик (декодер), как показано на рисунке 1.10.

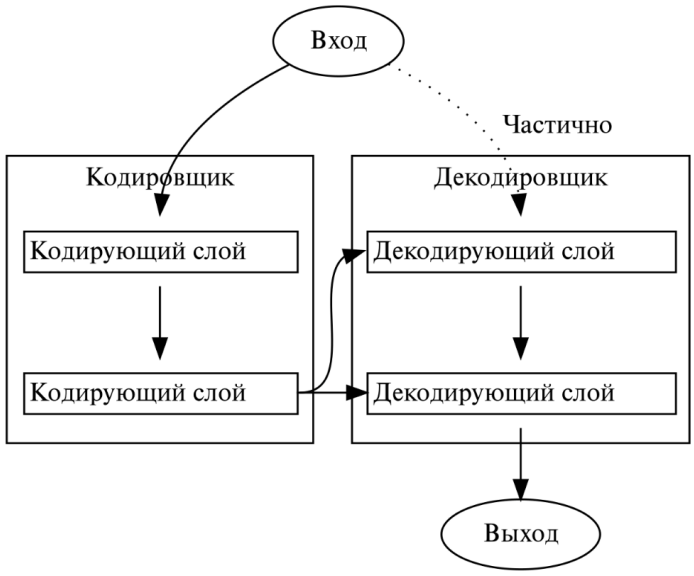

Рисунок 1.10 – Архитектура трансформера

На вход энкодера подается векторное представление последовательности, дополненное позиционными данными. В основе слоя декодера лежат методы и идеи, появившиеся в 2013 глубокого обучения, Word2Vec [11]. Они имееют другое название более распространённое – эмбеддинг [12]. Из набора последовательно берется пять слов, из них выбирается центральное и два слова до и после него – контекст центрального слова. Дальше сдвигаемся на одно слово и повторяем данный алгоритм. Это можно увидеть на рисунках 1.11 – 1.12.

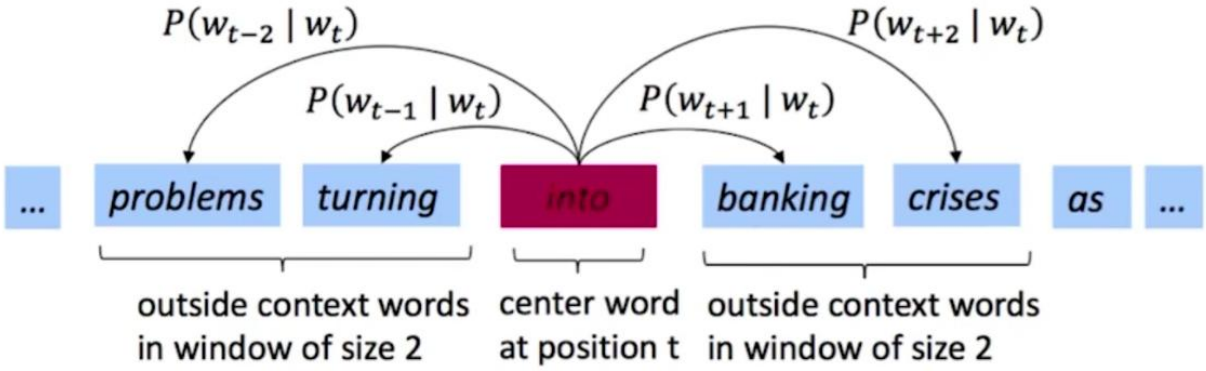

Рисунок 1.11 – Схематическое изображение алгоритма при центральном слове into

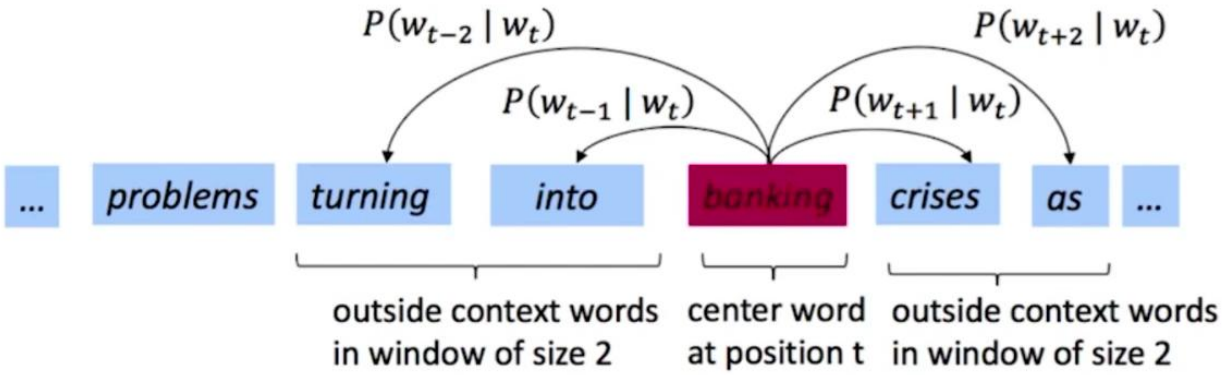

Рисунок 1.12 – Схематическое изображение алгоритма при центральном слове banking

Далее используется нейросеть, которая обучается на данных словах и позволяет сравнивать их смысл, контекст с помощью некоторой метрики, в основном применяется косинусное расстояние. Алгебраический вид, следующий:

$$similarity = \cos\theta = \frac{A \cdot B}{\|A\| \cdot \|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}} \quad (1.10)$$

где A и B – векторы.

Тогда, такие обученные слова в векторном виде будут называться эмбеддингом слов.

Так же необходимо учитывать контекст, поэтому воспользуемся алгоритмом выше, представленный на рисунках 1.11 – 1.12, когда у нас присутствует центральное слово и контекстом является два слова до и после него. На этих данных нейронная сеть будет учиться.

Вид данной нейросети будет показан на рисунке 1.13.

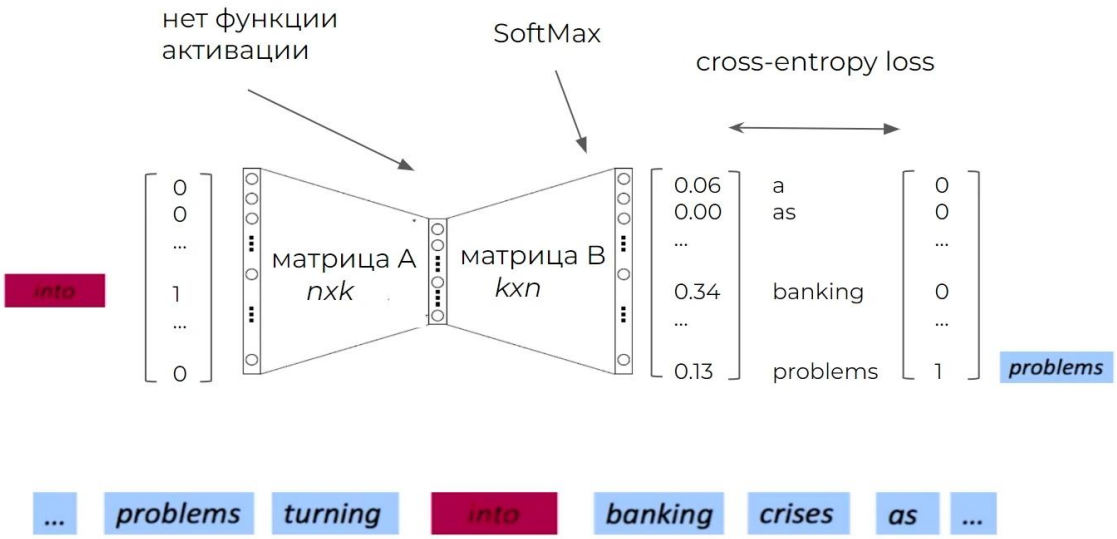

Рисунок 1.13 – Нейросеть модели Word2Vec

Как и было показано на рисунке 1.10 вход осуществляет через кодировщик. Сам энкодер имеет следующую схему на рисунке 1.14, где Word2Vec состоит в слое внимания.

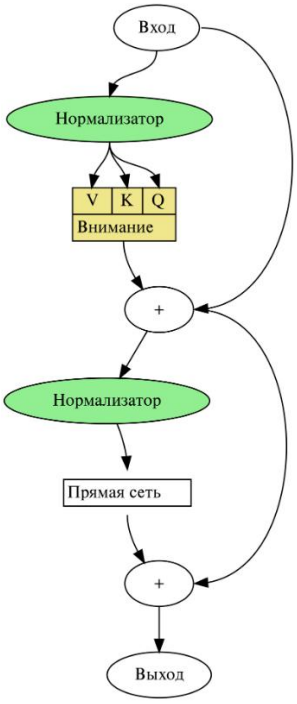

Рисунок 1.14 – Схема энкодера

Нормализатор. Применяется к входным данным перед слоем внимания. Стабилизирует обучение, уменьшая внутренний ковариационный сдвиг.

Слой внимания вычисляет зависимость между всеми элементами последовательности, алгебраический вид, следующий:

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \quad (1.11)$$

где Q – запрос;

K – ключ;

V – значение;

d_k – размерность ключей.

Между каждым переходом происходит сложение или остаточное соединение, к выходу слоя внимания добавится исходный вход, т.е. нормализатор.

Прямая сеть FFN. Состоит из двух слоев с активацией, добавляя нелинейность и преобразует признаки.

В конце происходит выход слоя. После FFN применяется остаточное соединение и нормализатор. Данный результат отправляется на декодировщик. Схема декодировщика представлена на рисунке 1.15

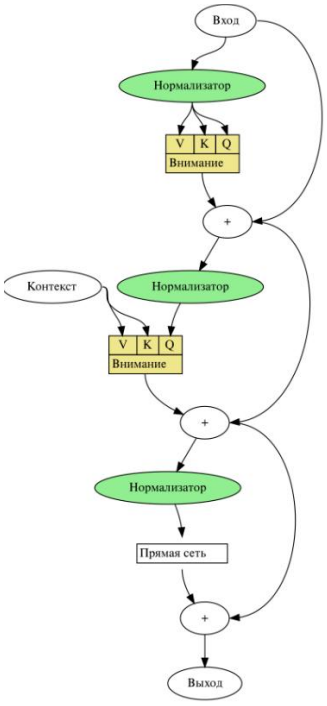

Рисунок 1.15 – Схема декодера

На вход декодера поступает результат, полученный на последнем шаге энкодера. Механизм нормализатора такой же, как и у декодера, поэтому опустим повторное описание. Оказавшись на слое самовнимания он будет являться маскировочным, и иметь следующий алгебраический вид:

$$MaskedAttention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}} + M\right)V \qquad (1.12)$$

где M – маска.

Маска используется для того, чтобы предотвратить утечку информации из будущих токенов в текущие.

Контекст от кодировщика. Выход последнего слоя кодировщика передаётся в декодировщик как ключи K и значения V для второго механизма внимания.

Остаточные этапы повторяются, как и в кодировщике. В итоге на выходе получаем вектор скрытого состояния, который проходит через линейный слой для преобразования в логиты – ненормированные оценки, чтобы генерировать их в выходные токены и поступает в Softmax [13], чтобы преобразовать логиты в вероятностное распределение hal словарем. Алгебраический вид, следующий:

$$P(w_i) = \frac{e^{z_i}}{\sum_{j=1}^{V} e^{z_i}} \qquad (1.13)$$

где z_i – логит для словаря w_i,

V – размер словаря.

Особенность Softmax заключается в температуре, которая управляет гладкостью и резкостью распределения вероятностей. Значение температуры принимает два диапазона значений:

– От нуля до единицы. Разница между вероятностями становится великой, что подходит для более точных ответом.

– Более единицы. Разница между вероятностями становится минимальной.

Несмотря на выдающиеся достижения трансформеров в глубоком обучении, их применение ограничено из-за квадратичной сложности вычислений, не позволяющей эффективно обрабатывать длинные последовательности свыше 10 000 токенов, что существенно сужает их применение для обработки масштабных данных.

На основе данной модели была разработана Mamba [14]. В отличие от трансформеров, Mamba использует принципиально иную архитектуру – SSM (State Space Models) [15]. Хотя SSM появились гораздо раньше, в глубоком обучении они долгое время не могли сравниться по эффективности с трансформерами, особенно в задачах языкового моделирования. Однако Mamba преодолевает это ограничение: её вычислительная сложность линейна, а пропускная способность в 5 раз выше, чем у классических трансформеров.

SSM применяется для характеристики скрытых состояний системы и прогнозирования их возможных изменений в ответ на входные воздействия. С математической точки зрения, она реализуется через систему из двух ключевых уравнений: уравнения состояния, описывающего эволюцию системы, и уравнения наблюдения, определяющего выходные величины. Алгебраический вид, следующий:

$$h'(t) = Ah(t) + Bx(t), \tag{1.14}$$

$$y(t) = Ch(t) + Dx(t) \tag{1.15}$$

где $h'(t)$ – производная состояния;

A – матрица перехода состояний;

B – матрица входа;

$x(t)$ – входной сигнал;

$h(t)$ – скрытое состояние;

$y(t)$ – выходное состояние;

C – матрица наблюдения;

D – матрица сквозной связи.

Схемы работы SSM представлены на рисунках 1.16 – 1.19.

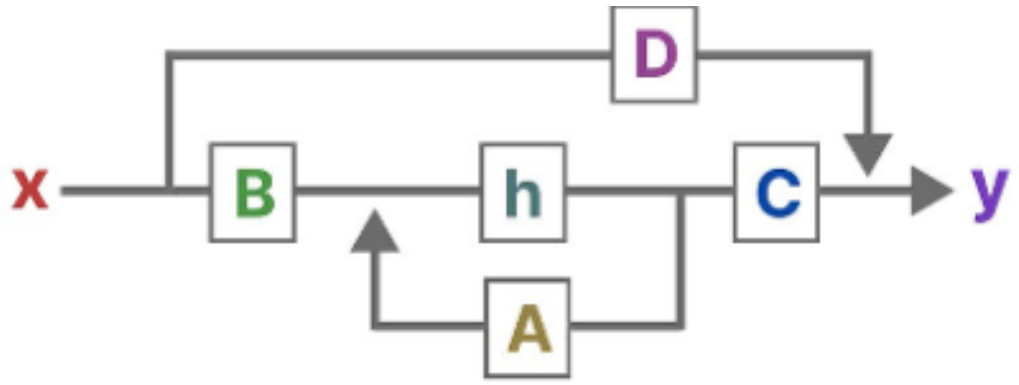

Рисунок 1.16 – Схема работы SSM

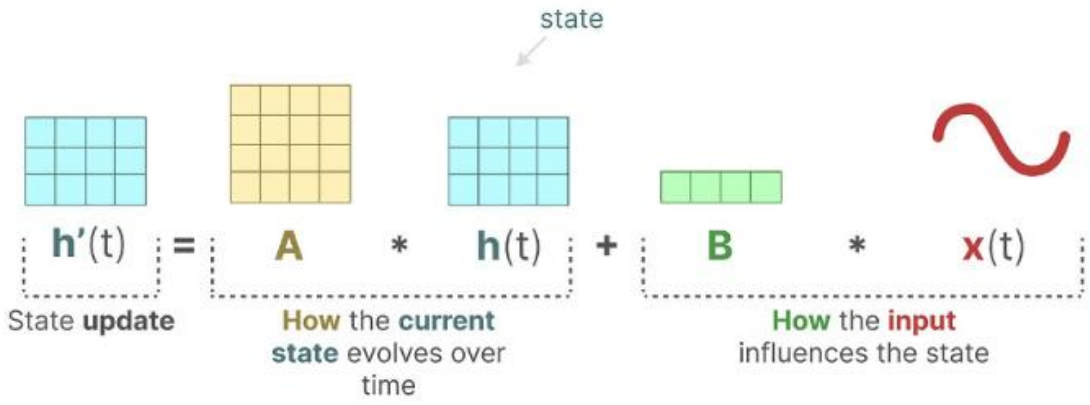

Рисунок 1.17 – Визуализация уравнение состояния

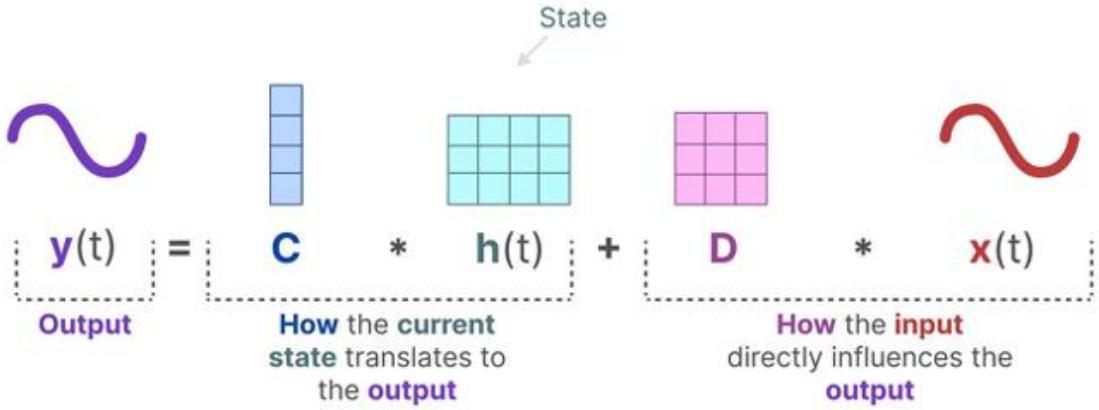

Рисунок 1.18 – Визуализация уравнение выхода

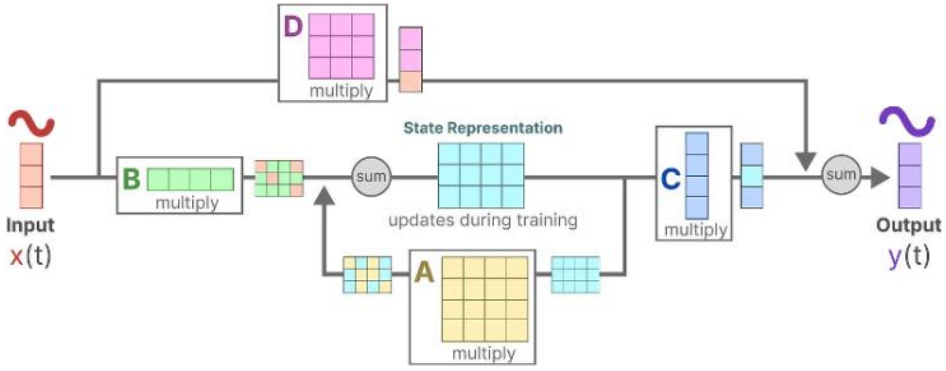

Рисунок 1.19 – Итоговая схема взаимодействия

Архитектура Mamba не отличается от трансформера, это можно увидеть на рисунке 1.20.

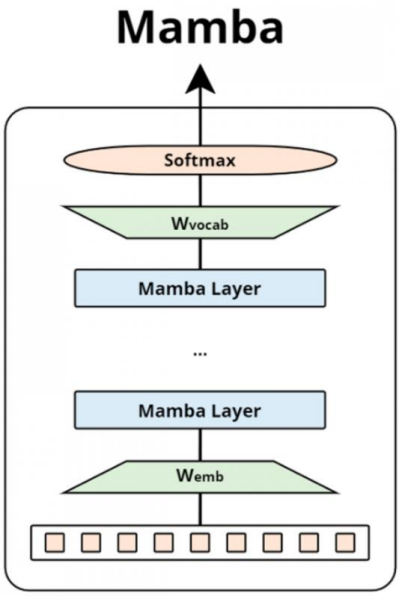

Рисунок 1.20 – Архитектура Mamba

Но по своему внутреннему строению слой Mamba отличается (рисунок 1.21).

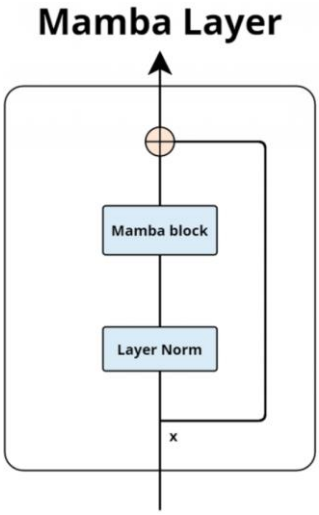

Рисунок 1.21 – Слой Mamba

Отличительной чертой является Mamba блок, в котором содержит SSM, как показано на рисунке 1.22.

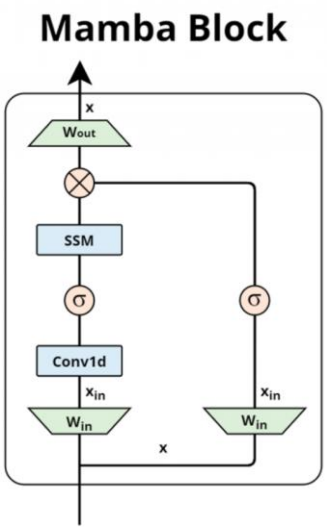

Рисунок 1.22 – Mamba блок

Блок использует модифицированную MLP-архитектуру с механизмом управления информационным потоком. Дополнительная ветвь, содержащая линейный слой и функцию активации, генерирует маску для поэлементного умножения, что позволяет избирательно усиливать или подавлять компоненты основного потока данных. Эта архитектура дополнена механизмом selective SSM, описанным ранее.

1.2.4 Выборка необходимых документов и данных

На данный момент, большинство общедоступных LLM при создании промпта ищут информацию в открытых интернет-источниках, но при поставленной цели и задачи необходимо искать среди тех данных, которые были предоставлены модели. Такой метод называется Retrieval Augmented Generation (RAG) [16] – это метод работы с большими языковыми моделями, когда пользователь пишет свой промпт, а система программно к этому вопросу «подмешивает» дополнительную информацию из каких-то внешних источников и подаете все целиком на вход языковой модели. Другими словами, добавление в контекст запроса к языковой модели дополнительную информацию, на основе которой языковая модель может дать пользователю более полный и точный ответ.

В текущем случае выборка будет происходить полностью из тех документов, которые находятся в базе данных корпорации.

На рисунке 1.23 отображена концепция RAG.

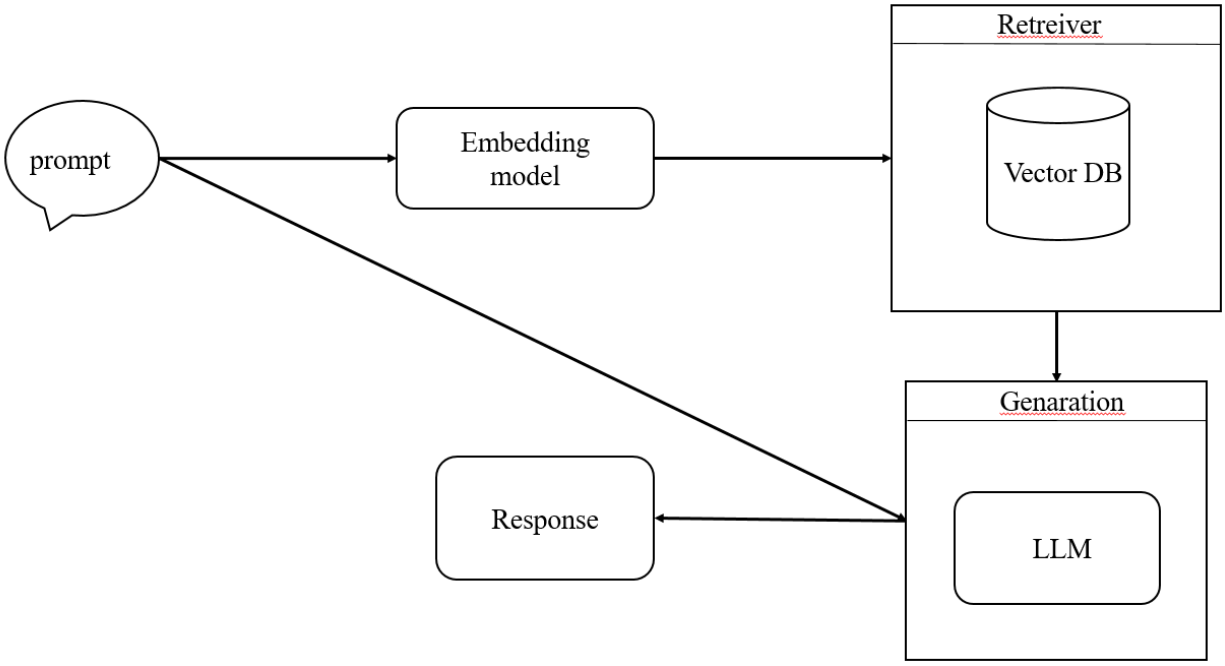

Рисунок 1.23 – Концепция RAG

Простыми словами, сотрудник вводит запрос, система его обрабатывает, ищет в корпоративных документах соответствующий документ, извлекает необходимую информацию, генерирующейся через языковую модель, и в конце возвращает ответ на поставленный запрос.

При внедрении данного метода могут возникнуть следующие проблемы:

1. Нечеткий поиск – просто взять запрос и найти по точному соответствию все куски из документа не получится.

2. Размер данных документа – неизвестен размер «кусков» текста, который обрабатывает LLM.

3. На заданный запрос может найти несколько релевантных документов.

Размер текста в контексте больших языковых моделей называется чанками – небольшие куски текста, которые между собой объединяются и получается осмысленный ответ. Тогда выделить следующие идею:

1. Чем меньше чанк по размеру, тем точнее будет буквальный поиск, чем больше размер чанка тем больше поиск приближается к смысловому.

2. Разные запросы пользователя могут содержать разное количество чанков, которое необходимо добавлять в контекст. Необходимо опытным путем подобрать тот самый коэффициент, ниже которого чанк смысла не имеет и будет лишь замусоривать ваш контекст.

3. Чанки должны перекрывать друг друга, чтобы был шанс подать на вход последовательность чанков, которые следуют друг за другом вместе, а не просто вырванные из контекста куски.

4. Начало и конец чанка должны быть осмысленными, в идеале должны совпадать с началом и концом предложения, а лучше абзаца, чтобы вся мысль была в чанке целиком.

К определению более релевантного документа можно воспользоваться методами из раздела 1.2.2:

1. TF-IDF (вычисляет важность каждого слова в документе относительно количества его употреблений в данном документе и во всей коллекции текстов).

2. BM25 (функция ранжирования, используемая поисковыми системами для упорядочивания документов по их релевантности данному поисковому запросу).

Если же документы, лежащие в базе знаний, не упорядочены между собой, то можно воспользоваться графом знаний. Граф знаний – это направленный реляционный граф, где вершины обозначают сущности, а рёбра отражают связи между ними. Такие графы строятся на основе триплетов вида (*субъект, отношение, объект*) или (s, r, o), что позволяет систематизировать информацию как о конкретных объектах, так и об абстрактных концепциях. Таким образом, граф знаний служит структурированной моделью для представления разнообразных фактов.

Таким образом документы можно структурировать между собой, обладая общей тематикой, что уменьшит время поиска и повысит точность. Пример такого графа представлен на рисунке 1.24.

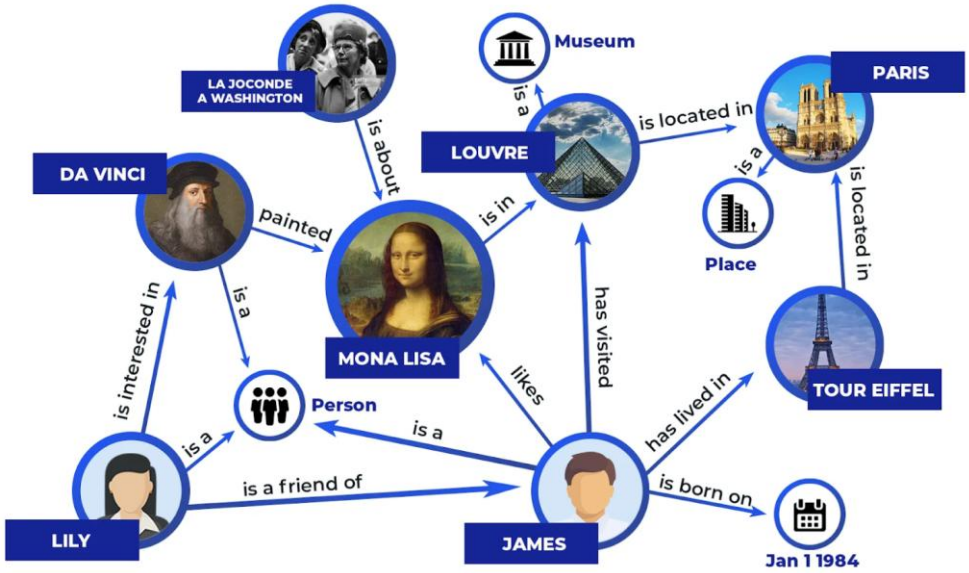

Рисунок 1.24 – Граф знаний

1.3 Формулирование требований

Обращая внимание на предметную область, были сформулированы определенные функциональные требования к разрабатываемой системе:

1. Пользователь должен иметь возможность авторизоваться для использования системы.

2. Система должна сформировать токен, с помощью которого пользователь может пользоваться внутренней API.

3. Пользователь должен иметь возможность формировать запрос.

4. Элементы интерфейса системы должны включать в себя поле ввода, чтобы пользователь мог сформировать запрос.

5. Система должна предоставлять возможность пользователю загружать необходимые файлы, если у него стоит задача найти необходимую информацию в них, если же это сотрудник, то внедрить/подключить к внутреннему корпоративному хранилищу.

6. Система должна формировать корректный и понятный ответ пользователю.

1.4 Обзор аналогов

Существует множество моделей, которые на данный момент могут удовлетворять условия актуальности и постановки задачи, но более мощные по своей точности, производительности и функционалу подходят следующие:

1. IBM Watson [17] – это мощная платформа искусственного интеллекта, которая включает в себя множество инструментов для обработки естественного языка, машинного обучения и аналитики данных. Она широко используется в корпоративных средах для автоматизации бизнес-процессов, улучшения клиентского обслуживания и оптимизации принятия решений. Watson может интегрироваться с корпоративной базой данных для создания чат-ботов, анализа текстов, прогнозирования и других задач, связанных с обработкой больших объемов данных.

2. Azure Cognitive Services от Microsoft [18] предлагает набор API-интерфейсов для интеграции функций искусственного интеллекта в приложения и системы. Эти сервисы включают обработку текста, речи, изображений и видео. С помощью Azure Cognitive Services можно создавать интеллектуальные приложения, которые взаимодействуют с корпоративными данными, анализируют тексты, распознают речь и многое другое.

3. Google Cloud AI [19] предоставляет инструменты и сервисы для разработки и развертывания моделей машинного обучения, включая Natural Language Processing (NLP). Платформа поддерживает создание и обучение собственных моделей, а также использование готовых решений. Google Cloud AI может быть использована для анализа текстов из корпоративных баз данных, автоматического ответа на запросы клиентов, классификации документов и других NLP-задач.

4. Amazon Lex [20] — это сервис для создания голосовых и текстовых интерфейсов на основе технологии Alexa. Он позволяет разрабатывать чат-боты и другие интерактивные интерфейсы, которые могут взаимодействовать с пользователями через текстовые сообщения или голосовые команды. Lex может быть интегрирован с корпоративной системой для автоматизации взаимодействия с клиентами, сотрудников или партнеров, обеспечивая быстрый доступ к необходимой информации.

5. OpenAI GPT [21] — это семейство языковых моделей, разработанных компанией OpenAI. Они используются для генерации текста, перевода, обобщения и других сложных задач обработки естественного языка. Хотя GPT изначально был создан для открытых систем, он может быть адаптирован и использован в корпоративных приложениях для различных целей, таких как автоматическое составление отчетов, ответы на вопросы пользователей и поддержка принятия решений на основе данных.

6. Hugging Face Chat [22] — это интерактивная платформа для общения с искусственным интеллектом. Она предназначена для тестирования и демонстрации возможностей различных языковых моделей, доступных на платформе Hugging Face Hub. Пользователи могут взаимодействовать с моделями прямо в браузере, задавая вопросы, проводя эксперименты и оценивая производительность моделей в реальных условиях.

7. DeepSeek [23] — это крупная языковая модель (LLM). Хотя модель пока находится на стадии раннего развития и менее известна широкой публике, она вызывает значительный интерес среди специалистов в области искусственного интеллекта благодаря своему инновационному подходу к обработке информации и широкому спектру потенциальных применений.

Следует отметить, что несмотря на наличие аналогов их использование невозможно в связи с санкциями и ограничением доступа российским пользователям. За счет того, что многие ограничены в мощностях программного обеспечения и проанализировав отечественные решения, отвечающих требованиям, была выбрана модель GigaChat от Сбербанка [24].

Для взаимодействия необходим доступ к API. Сбербанк предоставляет защищенное соединение для получения, чтобы данные нельзя было перехватить. С полным перечнем документов можно ознакомится в разделе GigaChat «Юридические документы GigaChat» [25].

Стоит отметить про локальное использование модели. Если же компании не подходит вариант с подключением к интернету или же у нее отсутствует ключ ПКЗИ для взаимодействия с GigaChat API, то необходимо на свои сервера скачать необходимую модель для функционирования системы.

2 ФУНКЦИОНАЛЬНЫЕ ВОЗМОЖНОСТИ И СПЕЦИФИКАЦИИ

2.1 Выбор модели

За внутреннею модель разрабатываемого программного комплекса была выбрана, выше упомянута, GigaChat. Данная модель включает в себя следующие необходимые компоненты:

1. Авторизация, для получения токена, которая предоставляет API для взаимодействия с моделью.
2. Добавление собственных файлов.
3. Обработка большого размера текста.

На данном этапе формируется список требований к функционалу и характеристикам программного комплекса, а также проводится анализ уже существующих решений. В ходе разработки были выявлены следующие требования:

1. Программный комплекс должен включать в себя возможность выбрать иную языковую модель, если в ходе доработок такие буду добавлены для иных задач.
2. Программный комплекс должен включать в себя возможность искать ответ не только внутри корпоративных документов, но и среди открытых источников. Сотрудник может получать ответ, не обращаясь к корпоративной базе данных.
3. Программный комплекс должен включать в себя возможность искать ответ в указанном файле. Сотрудник ранее работал с определенным файлом и знает, что в этом файле находится необходимая ему информация.

После того, как были определены цели был проведен анализ рисков, которые могут возникнуть при внедрении и использование программного комплекса. Выделенные следующие риски:

1. Удобство использования. Создание интуитивного, понятного интерфейса.

2. Совместимость. Обеспечить работоспособность на любую операционную систему.

Этап разработки обеспечивает реализацию функциональности на предыдущих этапах. Разработка велась поэтапно, в соответствие с предоставляемыми компонентами модели компонентами и требованиями программного комплекса, где каждый был реализован и протестирован отдельными функциями и модулями.

Тестирование производилось на каждом этапе для корректной, стабильной и полноценной работы программного комплекса.

2.2 Выбор инструментов реализации

В ходе анализа, основываясь на вышеопределенной области, был выбран набор используемых инструментов, который обеспечивает эффективную разработку. Также, одним из факторов выбора – популярность, стабильность, простота использования, функциональность. Рассмотрим данный набор инструментов разработки:

1. Jupyter Notebook [26] – это интерактивное веб-приложение для создания и совместного использования документов, содержащих код, визуализации данных, математические формулы и пояснительный текст. Это мощный инструмент для анализа данных, машинного обучения, численного моделирования и других задач, связанных с программированием и наукой о данных.

2. Visual Studio Code [27] – это бесплатный редактор исходного кода от компании Microsoft, который стал одним из самых популярных инструментов среди разработчиков благодаря своей гибкости, расширяемости и поддержке множества языков программирования. VS Code предоставляет мощные функции для редактирования кода, отладки, управления версиями и интеграции с различными системами контроля версий и платформами разработки.

3. Streamlit [28] – это библиотека для Python, которая позволяет быстро создавать веб-приложения, поддерживает различные виджеты, такие как ползунки, выпадающие списки, кнопки и чекбоксы, что позволяет пользователям взаимодействовать с приложением в реальном времени. Она идеально подходит для создания прототипов, чатов, ботов, дашбордов и интерактивных отчетов, позволяя разработчикам сосредоточиться на логике приложения, а не на фронтенде.

4. UV был выбран в качестве альтернативного менеджера зависимостей и сборщика Python-пакетов. Этот инструмент сочетает в себе высокую скорость работы, современный подход к управлению зависимостями и совместимость с экосистемой Python [29]. UV обеспечивает быстрое разрешение зависимостей, параллельную установку пакетов и поддержку виртуальных окружений, что делает его отличным выбором для больших проектов. Кроме того, UV совместим с существующими стандартами, такими как pyproject.toml и requirements.txt, что упрощает миграцию с других инструментов. Он также поддерживает создание и публикацию пакетов, помогая разработчикам поддерживать порядок в проекте и ускорять рабочие процессы. Благодаря своей производительности и удобству [30].

Выбранный список инструментов описывает функциональность, удобство, гибкость, скорость и удобство пользования. Обеспечивают полную поддержку для всех этапов разработки, начиная от настройки конфигурации до анализа и редактирования полученного ответа

2.3 Определение входных и выходных данных

2.3.1 Входные данные

Для полного функционирования программного комплекса, обеспечивающая ответ на запрос, были определены следующие данные, которые поступают в нее:

1. Авторизационные данные, предоставляющие токен для API модели.

2. Сформированный корректный запрос.
3. Загруженные документы.
4. Выбор определенного документа для поиска информации при необходимости.
5. Выбор языковой модели при необходимости.

2.3.2 Выходные данные

Выходные данные включают в себя ответ на сформированный запрос, который представляет собой отредактированный, отформатированный текст.

3 ПРОЕКТИРОВАНИЕ

В данной работе использован системный подход к проектированию программного комплекса. Раздел включает в себя описание диаграммы прецедентов, последовательности, пакетов. Для структурирования и визуализации компонент программного комплекса была выбрана нотация UML (Unified Modeling Language).

Для определения и описания ключевых функций программного комплекса была построена диаграмма прецедентов. Чтобы показать, как объекты системы взаимодействуют друг с другом при выполнении основных процессов, были разработаны диаграммы последовательности. Для того, чтобы отобразить связь между основными папки проекта была спроектирована диаграмма пакетов.

3.1 Диаграмма прецендентов

Диаграмма прецедентов, представленная на рисунке А.1 в приложении А, описывает функциональные возможности программного комплекса, а также взаимодействие пользователя с программным комплексом. Рассмотрим каждый элемент диаграммы и его роль:

1. Создание запроса позволяет отправить запрос в систему для обработки и выдачи релевантного ответа. До этого этапа можно дополнить следующим функционалом:

   a. Выбор языковой модели, предоставляет сотруднику возможность выбрать модель из списка предоставленных для обработки запроса в необходимом формате.

   b. Область поиска, предоставляет сотруднику возможность выбрать вариант поиска в системе.

   c. Поиск из корпоративных документов, предоставляет сотруднику релевантный ответ из извлеченных корпоративных документов.

d. Поиск из открытых источников, предоставляет сотруднику релевантный ответ из источников общего доступа, интернет.

e. Добавление документа к запросу, позволяет сотруднику искать релевантный ответ из приложенного документа.

2. Загрузка документов:

a. Просмотр корпоративных документов, позволяет сотруднику получить список всех имеющихся документов.

b. Удаление документов, позволяет сотруднику удалять загруженные собственные документы. У сотрудника будет возможность удалить выбранные документы, так и все сразу.

В качестве основного актора выступает сотрудник, который взаимодействует с программным комплексом. Сотрудник может ввести запрос, выбрать необходимую языковую модель, по умолчанию ответ ищется в корпоративных документах, но можно выбрать генерацию ответа из приложенных документов или открытых источников, при необходимости можно загрузить необходимые документы, просмотреть и удалить их.

Далее были описаны основные взаимодействия, предусмотренные в данной системе:

– Сотрудник вводит запрос, после чего он может выбрать дополнительные параметры, такие как: выбрать языковую модель, источник получения.

– Сотрудник выбирает языковую модель, по умолчанию установлена GigaChat.

– Сотрудник выбирает источник поиска ответа, по умолчанию поиск происходит из извлеченных корпоративных документов, так же существуют варианты поиска в открытых источниках и выбор документов из корпоративной базы и загруженных лично сотрудником.

– Сотрудник может загружать имеющиеся документы при необходимости.

– Сотрудник может просмотреть документы, находящиеся в корпоративной базе и загруженных самим сотрудником.

– Сотрудник может удалить собственные загруженные документы выборочно либо все.

– По завершению выбора необходимых параметров сотрудник отправляет запрос, и система возвращает релевантный ответ.

В данном подразделе была представлена и проанализирована диаграмма прецедентов системы. Диаграмма отразила основные сценарии использования, такие как создание запроса, выбор языковой модели, выбор источника поиска, добавление документов, удаление добавленных документов.

3.2 Диаграмма последовательности

Диаграммы последовательности, представленные на рисунках Б.1 – Б.2 приложения Б, были спроектированы компоненты системы во временном контексте. В данном подразделе будут рассмотрены диаграммы последовательности, включающая такие объекты, как сотрудник, пользовательский интерфейс (UI), GigaChat API.

Взаимодействие начинается с попадания на главный экран системы, где происходит получение токена и документов из корпоративной базы и те, которые были до этого загружены сотрудником.

Далее вводится запроса. На UI отображены параметры такие как: выбор языковой модели, выбор источника поиска. При необходимости сотрудник может изменить данные параметры.

После определения параметров сотрудник отправляет запрос, UI обрабатывает и передает в GigaChat API. API формирует релевантный ответ и возвращает обратно в UI, где происходит отображение данного ответа.

Для того, чтобы взаимодействовать с документами необходимо их добавить. После можно проверить их успешную загрузку в базу. В

дальнейшем, если файлы устарели, необходимо их обновить или очистить базу полностью.

Таким образом, диаграммы последовательностей демонстрируют, как сотрудник, взаимодействуя с интерфейсом, управляет процессом подготовки данных, настройки параметров, и как различные компоненты системы координируют свои действия для достижения цели.

3.3 Макет интерфейса

В данном подразделе был рассмотрен макет пользовательского интерфейса, представленный на рисунках В.1 – В.4 приложения В. Одна из главных задач – организация элементов интерфейса, которая должна быть интуитивно понятна сотрудникам для взаимодействия с системой.

Интерфейс включается в себя левую боковую панель, которую можно сворачивать при необходимости, и основную центральную рабочую область. Боковая панель предназначена для перехода между страницами разделов.

В ходе разработки было принято решение разделить все страницы на три основных раздела: «Описание», «ПКПвКБЗ», «Документы». Раздел «Описание» включается в себя страницу «Главная» – страница, на которую попадает сотрудник при запуске системы, где отображено описание созданной системы. Раздел «ПКПвКБЗ», который расшифровывается как «Программный комплекс интеллектуального поиска в корпоративных документах» включается в себя следующие страницы: «Чат», «Закрыть чат». На странице «Чат» сотрудник задает свой вопрос, и система выдает ответ. Страница «Закрыть чат» служит для окончание работы с системой. В разделе «Документы» так же было создано две страницы: «Загрузка документов», «Просмотр/удаление документов». На странице «Загрузка документов» отображен виджет для загрузки файлов, который принимает выбранные документы сотрудника и загружает их в хранилище. И на странице «Просмотр/удаление документов» отображен список текущих документов,

которые были взяты из корпоративной базы и личного хранилища сотрудника. Те документы, которые находятся в корпоративной базе не могут быть никак изменены, а те документы, которые были загружены в личное хранилище сотрудником, могут удаляться им же при необходимости.

Цветовая схема интерфейса выбирается от параметров системы компьютера, т.е. если у сотрудника светлая тема, то и схема то же будет светлая, в противном случае – темная.

Дополнительные функциональные элементы включаются в себя подсказки. Они могут быть следующего характера:

– При наведении курсором на виджет всплывает краткая информация использования.

– На самом виджете отображена информация пользования.

Данные справки помогают сотруднику быстро ориентироваться в настройках системы. Как и было сказано выше, при переходе на страницу «Главная» будут описаны возможности корректного взаимодействия с системой.

3.4 Диаграмма пакетов

В данном подразделе рассматривается диаграмма пакетов, разработанная с использованием UML.

Всего была разработано четыре основных пакета:

– Пакет Config. Конфигурационный пакет, подключающий необходимые фреймворки и библиотеки, а также переменные окружения.

– Пакет ChatSettings представляет собой интерфейс чата с необходимыми компонентами и настройками импортирующиеся из пакета Config.

– Пакет DocsSettings нужен для того, чтобы добавлять документы в хранилища, просматривать данное хранилище и удалять документы, которые потеряли актуальность или требуют обновление до новой версии. Само

подключение к хранилищу и корпоративной базе импортируется из пакета Config.

– Пакет App является центральным пакетом, который собирает вышеперечисленные пакеты для полной навигации между собой, на данном этапе была составлена диаграмма пакетов, представленная на рисунке Г.1 приложения Г, демонстрирующая связь между ними.

4 КОДИРОВАНИЕ

4.1 Конфигурация инструментов разработки

При разработке программного комплекса была выбрана операционная система Windows. Установка необходимых инструментов, сред и пакетов в основном происходили через ссылки на официальные источники, где находятся установочные устройства. Но для более детальной настройки был установлен пакетный менеджер chocolatey, была выполнена следующая команда:

@"%SystemRoot%\System32\WindowsPowerShell\v1.0\powershell.exe" -NoProfile -InputFormat None -ExecutionPolicy Bypass -Command "iex ((New-Object System.Net.WebClient).DownloadString('https://chocolatey.org/install.ps1')) " && SET "PATH=%PATH%;%ALLUSERSPROFILE%\chocolatey\bin"

Далее, для установки Visual Studio Code с помощью chocolatey была выполнена следующая команда:

choco install vscode

Теперь, необходимо установить Python, версия при установке была последняя 3.12, была выполнена следующая команда:

choco install -y python3

Для корректной работы с Python в VS Code необходимо установить расширения зависимостей для полноценного поддержания кода. Для этого были выполнены следующие команды:

code --install-extension ms-python.python

code --install-extension ms-python.vscode-pylance

После того, как была произведена успешная установка рабочей среды, необходимо создать проект с управлением в нем зависимостей. В качестве фреймворка был выбран UV. Для его установки была выполнена следующая команда:

pip install uv

Переходим к созданию проекта. Название было выбрано стандартное venv, для создания рабочей области проекта используется следующая команда: uv init venv. По мере расширения проекта в файл pyproject.toml добавляются библиотеки, устанавливаемые следующей командой:

uv add package_name

С актуальной версией файла pyproject.toml можно ознакомиться в листинге Д.1 приложения Д.

Разберем библиотеки, которые использовались в ходе разработки программного комплекса.

Для отображения всех необходимых элементов интерфейса UI была выбрана библиотека streamlit, ее установка была выполнена следующей командой:

uv add streamlit

Для обмена данными с сервисов GigaChat API использовалась библиотека requests, ее установка была выполнена следующей командой:

uv add requests

Для корректного формирования данных из корпоративной базы и документов была использована библиотека pandas, ее установка была выполнена следующей командой:

uv add pandas

Для того, чтобы получать необходимые данные из корпоративной базы, которая представляет собой реляционную базу данных, была использована библиотека psycopg2, ее установка была выполнена следующей командой:

uv add psycopg2

Для того, чтобы обрабатывать данные из документа формата docx была использована библиотека python-docx, ее установка была выполнена следующей командой:

uv add python-docx

Для того, чтобы обрабатывать данные из документа формата pptx была использована библиотека python-pptx, ее установка была выполнена следующей командой:

uv add python-pptx

Для того, чтобы обрабатывать данные из документа формата pdf была использована библиотека pypdf, ее установка была выполнена следующей командой:

uv add pypdf

Для того, чтобы обрабатывать и суммировать большой объем текста были использованы комплекс библиотек: langchain, langchain-community, langchain-core, langchain-text-splitters, их установка была выполнена следующей командой:

uv add langchain langchain-community langchain-core langchain-text-splitters

Для того, чтобы загрузить обработанный и суммированный текст в модель были использованы две библиотеки: langchain-gigachat, langchain-ollama. GigaChat предназначена для удаленного взаимодействия, Ollama для локального обращения к модели, их установка была выполнена следующей командой:

uv add langchain-gigachat langchain-ollama

Сразу после установки streamlit необходимо настроить secrets.toml расположенный в папке .streamlit – папка окружения streamlit. В ней необходимо оставить данные удаленного взаимодействия с GigaChat API и Корпоративной базой, которая представляет собой реляционную базу данных. Итоговая конфигурация представлена в листинге Д.2 приложения Д.

Данные этапы позволят создать полноценную работоспособную срезу для разработки описываемого программного комплекса.

4.2 Программная реализация

В созданном проекте было реализовано четыре пакета: app, config, chat_settings, docs_settings.

В каждом пакете для дополнительного функционала для корректной работоспособности в различных частях системы, был создан файл utils.py

Пакет config предназначен для размещения библиотек, которые были описаны в разделе 4.1 и дополнительных функций конфигурации проекта, такие как: get_gigachat_token, get_files_name, get_name_files_to_id, get_models, sql_cursor, get_knowledge. Все они служат для инициализации и получения данных при попадании на главную страницу системы.

Пакет chat_settings служит для обмена сообщениями между сотрудником, задающий запрос, и внутренней моделью, которая обрабатывает запрос и выдает ответ. Так же пакет имеет визуальную функциональную составляющую в файле chat.py.

Пакет docs_settings содержит модули для работы с корпоративной базой и документами. Модули LoadBase, UploadFiles, ViewFiles представляют собой функциональность, направленную на работу с документами, которые сотрудник может добавлять или удалять, а также просматривать их и то, что находится в корпоративной базе. Саму базу сотрудник изменять не может. Корпоративная база получается путем выгрузки разрешенных данных, предварительно обговоренной иными лицами корпорации связанных с информационной безопасностью.

Пакет app является центральным пакетом, где собираются вышеописанные пакеты для логической координации между собой и взаимодействия. Модуль Main представляет собой отображения страниц системы и их структуру. Модуль App является центральным модулем, который наследует функции из пакета config для создания сессии, чтобы данные, при попадании на какую-либо страницу, не требовали повторного извлечения.

4.3 Руководство администратора

Для того, чтобы пользоваться системой необходимо ее развернуть, в данном случае подразумевается, что у корпораций существует лицо, которое занимается данной задачей и после разворачивания интегрирует систему в корпоративный источник.

Для корректной работы программного комплекса, если модель расположенная удалена, требуются следующие системные характеристики:

– Операционная система: Windows 10 и выше, дистрибутивы ядра Linux, MacOS.

– Центральный процессов: Intel i5 и выше, AMD Ryzen 3 и выше.

– Оперативная память: минимум 8 ГБ ОЗУ и выше.

– Графический процесс: NVIDIA 1050 и выше, если центральный процессов имеет графический процессор, то он в нем может не нуждаться.

– Память компьютера: минимум 30 ГБ для установки необходимых приложений.

– Подключение к интернету для отправки запросов на удаленную модель GigaChat API.

Если же модель будет разворачиваться на личных серверах корпорации, требует следующие системные характеристики:

– Операционная система: Windows 10 и выше, дистрибутивы ядра Linux, MacOS.

– Центральный процессов: Intel Xeon и выше.

– Оперативная память: 64 ГБ ОЗУ и выше.

– Графический процесс: NVIDIA RTX 3090 и выше.

– Память компьютера: 500 ГБ и выше, если необходимо использовать большее количество моделей.

Администратор может модернизировать код программы для расширения потребностей системы и данное программное обеспечение является открытым. Для этого потребуется любая доступная среда разработки.

Процесс клонирования и установки репозитория выглядит следующим образом:

1. Открыть терминал и клонировать репозиторий с GitHub: git clone https://github.com/egorbeckish/Intelligent-Search-System.
2. Перейти в папку проекта: cd Intelligent-Search-System.
3. Установка менеджера зависимостей UV: pip install uv.
4. В папке проекта необходимо установить зависимости, указанные в файле pyproject.toml: uv pip install -r path/pyproject.toml.
5. Если же модель, расположенная удаленно, то необходимо перейти на платформу GigaChat, сгенерировать необходимые API ключи для взаимодействия.
6. Сгенерированные API ключи вставить в файл secrets.toml.

После полноценной инициализации, администратор может запустить систему для проверки работоспособности: uv run streamlit run app.py

При корректном запуске, администратор может приступить к модернизации программного кода. Необходимая инструкция по установке и информации системы размещена в репозитории на GitHub в файле README.md.

4.4 Руководство сотрудника

В данном разделе будет описаны основные функции системы для корректной работы сотрудника.

1. Панель перенаправления:
    a. Главная – страница описания о «Программный комплекс интеллектуального поиска в корпоративных документах». На ней происходит инициализация необходимых данных, как показано на рисунке Е.1 приложения Е.
    b. Чат – страница с чатом системы.
    c. Закрыть чат – выход из системы.

d. Загрузка документов – страница добавление документов в хранилище.

e. Просмотр/удаление документов – страница просмотра загруженных документов и находящихся в корпоративной базе. Панель представлена на рисунке Е.2 приложения Д со всеми вышеперечисленными страницами.

2. Работа с чатом:

a. Выбрать языковую модель из предоставленного списка. По умолчанию выбрана GigaChat.

b. Выбрать область поиска ответа. По умолчанию стоит выборка из корпоративной базы. Так же предоставлены варианты: выбрать определенный документ, поиск из открытых источников. Краткая информация об вариантах разворачивается при наведении на них, как показано на рисунке Е.3 приложения Е.

c. Очистить чат. При переполненном чате можно его очистить, если та информация, которая была отображена не используется.

3. Работа с загрузкой документов:

a. Выбрать документы, хранящиеся в папке сотрудника. Для верной загрузки на виджете отображена подсказка с доступными форматами и максимальным размером документа, как показано на рисунке Е.4 приложения Е. После, при нажатии на виджет появляется интерфейс операционной системы, где показывается папка с файлами, представлено на рисунке Е.5 приложения Е.

4. Просмотр/удаление документов:

a. Отображается список документов сотрудника и выгруженные из корпоративной базы.

b. Удаление выбранных файлов из хранилища сотрудника.

c. Удаление всех документов их хранилища сотрудника.

Следуя данной инструкции, сотрудник сможет эффективно взаимодействовать с системой.

5 ТЕСТИРОВАНИЕ

Тестирование – это ключевой этап в процессе разработки, который представляет собой значимый этап разработки, целью которого служит оценка правильности функционирования программы, обнаружение ошибок и недостатков, а также убедиться в её соответствии заданным требованиям. Этот раздел рассматривает применение методов тестирования, порядок выполнения, примеры тестовых случаев и итоговые данные. Тестирование проводилось вручную, что дало возможность детально изучить работу системы и своевременно выявлять возникающие проблемы. Детальное описание методологии, процедур и тестовых сценариев поможет обеспечить повторяемость и эффективность тестирования в будущих этапах доработки и совершенствования системы.

5.1 Методология тестирования

Как и было выше сказано, для тестирования программного обеспечения, разработанного программного комплекса, используется ручная методология тестирования. Данный метод не включал в себя инструменты автоматизированного тестирования. Ниже перечислены факторы выбора ручного тестирования:

1. За счет того, что данной системой будет пользоваться человек, а в частности сотрудник компании, то ручная методология позволит наиболее понятно выявить ошибки, затруднения, с которыми в последствие можно будет столкнуться. Этот вариант содержит в себе интуитивность и понятность интерфейса, удобность пользования системы в целом.
2. На данном этапе разработке, как и было сказано в разделе 1.4 за счет ограничения в ресурсах программного обеспечения, поддержка автоматизированных тестов была нецелесообразна.

В данном методе ручного тестирования были использованы следующие этапы:

– В заявленном макетном интерфейсе проводилось тестирование корректности отображения всех необходимых визуальных элементов, навигацию по системе.

– Для заявленных функциональных возможностей была проведена проверка соответствий. Тестирование основного функционала, такие как: создание запроса с необходимыми параметрами поиска в корпоративной базе, добавление документов сотрудника в хранилище, просмотр и удаление документов из хранилища, выход из системы.

Применение ручного метода к тестированию позволило добиться высокого уровня качества и надежности программного продукта, оперативно обнаруживать и исправлять недостатки, а также повысить удобство использования системы для сотрудников.

5.2 Процедура тестирования

Процедура тестирование программного обеспечения программного комплекса также выполнялась вручную. В процессе тестирования особое внимание уделялось релевантности ответа в разных размерах и форматах документов, которые обрабатывали языковые модели, с учетом дополнительных параметров.

Целью тестирования необходимо было убедится, что используемые языковые модели, такие как: GigaChat, Llama 3.1, Llama 3.2, находят необходимый документ и выдают релевантную информацию, которая соответствует задаваемому запросу в данном документе. Так же подавались документы, в которых было явно понятно какая информацию в нем присутствует и нужно было получить только ответ на запрос. И проверялись открытые источники для обнаружения полноты информации, которую языковые модели могут выдать.

Так же, проверялась работа с документами, которые сотрудник может добавить в хранилище, просмотреть его и удалить при необходимости.

5.3 Сценарии тестирования

Сценарии тестирования включают в себя проверки корректной работы программного комплекса, как языковые ищут информацию из открытых источников и заканчивая тем, как они определяют необходимый документ из корпоративной базы и находят в данном документе релевантный ответ на запрос. Так же будет рассмотрена загрузка, просмотр и удаление из хранилища сотрудника. Каждый сценарий описывается определенной последовательностью действия, которую необходимо соблюдать для того, чтобы не возникало ошибок входи использования системы.

Первый сценарий начинается с попадания на страницу чата, где необходимо выбрать языковую модель, которая будет использоваться для поиска.

Вторым сценарием необходимо выбрать зону поиска, то есть откуда языковая модель будет извлекать необходимую информацию.

Третьем сценарием следует сформировать свой запрос и передать его в чат, который в свою очередь отправит данные из первых двух сценариев, включая третий, на языковую модель, где в последующем после обработки вернется ответ.

Четвертый сценарий включается в себя добавление документов в хранилища с соответствующим форматом и размером.

Пятый сценарий представляет собой просмотр актуальной корпоративной базы с хранилищем и удаление из собственного хранилища документов, которые устарели или имеют более актуальную версию.

5.4 Результаты тестирования

Во время проведения тестирования основная задача ставилась на то, чтобы ответ языковых моделей был релевантен задаваемому запросу, т.е. контекст ответа не отличался от контекста запроса. И, конечно, было

проверена работа с документами. В соответствие с вышесказанным были получены следующие результата тестирования:

1. Модель llama3.2:3b и зона поиска:

– Открытые источники. За счет того, что модель имеет минимальные входные параметры из-за ограничения программного обеспечения ответ не всегда точный или же имеет дефекты, связанные с форматированием, кириллицей, что наблюдается на рисунке Ж.1 приложения Ж.

– Файл, в котором содержится информацию о необходимых требованиях написания отчета. Ответ модели представлен на рисунке Ж.2 приложения Ж.

– Корпоративная база. Система ищет по заданному запросу сначала релевантный файл, а потом в данном файл информацию, как это представлено на рисунке Ж.3 приложения Ж.

2. Модель llama3.1:8b и зона поиска:

– Открытые источники. Данная модель имеет в разы больше входных параметров, что повышает точность и куда лучше отформатированный ответ, как показано на рисунке Ж.4 приложения Ж.

– Файл, в котором содержится информацию о необходимых требованиях написания отчета. Ответ модели представлен на рисунке Ж.5 приложения Ж.

– Корпоративная база. Система ищет по заданному запросу сначала релевантный файл, а потом в данном файл информацию, как это представлено на рисунке Ж.6 приложения Ж.

3. Модель GigaChat и зона поиска:

– Открытые источники. Данная модель является одной из мощной в сфере языковых моделей, полученный ответ наполненный и корректно структурирован, как показано на рисунке Ж.7 приложения Ж.

– Файл, в котором содержится информацию о необходимых требованиях написания отчета. Ответ модели представлен на рисунке Ж.8 приложения Ж.

– Корпоративная база. Система ищет по заданному запросу сначала релевантный файл, а потом в данном файл информацию, как это представлено на рисунке Ж.9 приложения Ж.

4. Загрузка документов в хранилище предоставляет интерфейс операционной системы, где выбирается папка с документами, где хранит их сотрудник, далее их выбирают и происходит их загрузка, в конце появляется уведомление о том, что документы попали в хранилище, как показано на рисунках Ж.10 – Ж.11 приложения Ж.

5. Просмотр и удаление документов. После того, как документы были загружены необходимо убедиться в их корректном отображение и что их можно использовать, на рисунке Ж.12 приложения Ж отображен данный результат. Как можно было заметить, там же отображаются документы, хранящиеся в корпоративной базе, на которые сотрудник никак повлиять не может, т.е. при попытки удалить документы, удалятся те, которые были загружены сотрудником, как показано на рисунке Ж.13 приложения Ж.

По итогу тестирования программного комплекса, каждая модель выдала результат, соответствующий ожиданием. Это показатель того, что разработанный программный комплекс, внедренный в корпоративную базу, заметно увеличивает работоспособность с документами, что подтверждает эффективность выбранных моделей.

Заключение

Выполнение выпускной квалификационной работы позволило достичь всех поставленных целей и решить поставленные задачи, а также закрепить профессиональные компетенции, приобретённые в течение всего периода учёбы.

В результате проведения анализа требований были выявлены проблемные аспекты, появляющиеся в процессе обработки большего потока информации, хранящиеся в документах, а также был проведен краткий экскурс в сферу обработки естественного языка, векторного представления слов и актуальных видов языковых моделей для современных решений. Помимо этого, в рамках анализа были сформулированы функциональные требования и проведен обзор аналогов, который также показал актуальность разработки в связи с санкциями и ограничением доступа российским пользователям.

Далее были определены спецификации системы, входные и выходные данные. Здесь же был приведен обзор выбранных инструментов разработки, включающий традиционный стек специалиста по работе с данными: Visual Studio Code, Python, UV, Streamlit.

Далее последовал этап проектирования, включающий разработку диаграмм прецедентов, последовательности, а также создание макетов интерфейса.

На стадии кодирования были воплощены все разработанные ранее диаграммы прецендентов и последовательности. Дополнительно созданы инструкции для администратора и сотрудников, проведена настройка инструментов разработки, что в совокупности обеспечивает полную документацию по установке и запуску системы.

В конце каждого этапа разработки проводилось тестирование программы для полного функционала реализованных модулей. Особое внимание уделялось как выбранные языковые модели будут обрабатывать

запрос и искать в указанном источнике необходимую информацию. В итоговом результате модели показали корректную и стабильную работу на примере работы с подключенной корпоративной базой и выбранных документов.

В дальнейшем программный комплекс можно интегрировать в корпорации с большим объемом документооборота для увеличения производительности бизнес-процессов.

Следующие планы по развитию программного комплекса заключаются в том, чтобы увеличить функционал системы такой как: обработка иных форматов документов, улучшение пользовательского интерфейса, просмотр содержимого документа в самом программном комплексе.

Список использованных источников

1. Christopher D. Manning Human Language Understanding & Reasoning [Электронный ресурс]: сайт American Academy of Arts and Sciences. URL: https://www.amacad.org/sites/default/files/publication/downloads/Daedalus_Sp22_09_Manning.pdf (дата обращения: 04.06.2025).
2. ChatGPT [Электронный ресурс]: сайт OpenAI. URL: https://openai.com/stories/ (дата обращения: 04.06.2025).
3. Алан М. Тьюринг МОЖЕТ ЛИ МАШИНА МЫСЛИТЬ? [Электронный ресурс]: сайт Etheroneph. URL: https://www.etheroneph.com/files/can_the_machine_think.pdf (дата обращения: 04.06.2025).
4. Jamell Ivor Samuels One-Hot Encoding and Two-Hot Encoding: AnIntroduction [Электронный ресурс]: сайт ResearchGate. URL: https://www.researchgate.net/publication/377159812_One-Hot_Encoding_and_-Two-Hot_Encoding_An_Introduction (дата обращение: 04.06.2025).
5. Neil Christian R. Riego, Danny Bell Villarba Utilization of Multinomial Naive Bayes Algorithm and Term Frequency Inverse Document Frequency [Электронный ресурс]: сайт arXiv. URL: https://arxiv.org/pdf/2306.00018 (дата обращение: 04.06.2025).
6. Introduction to Information Retrieval [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/class/cs276/handouts/lecture12-bm25etc.pdf (дата обращение: 04.06.2025).
7. N-gram Language Models [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/~jurafsky/slp3/3.pdf (дата обращение: 04.06.2025).
8. RNN [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/~jurafsky/slp3/8.pdf (дата обращение: 04.06.2025).
9. LSTM [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/~jurafsky/slp3/8.pdf (дата обращение: 04.06.2025).

10. The Transformer [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/~jurafsky/slp3/9.pdf (дата обращение: 04.06.2025).

11. Introduction and Word2Vec [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/class/cs224n/readings/cs224n_winter2023_lecture1_notes_draft.pdf (дата обращение: 04.06.2025).

12. Vector Semanticsand Embeddings [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/~jurafsky/slp3/6.pdf (дата обращение: 04.06.2025).

13. Michael Franke, Judith Degen The softmax function: Properties, motivation, and interpretation[Электронный ресурс]: сайт Stanford. URL: https://alpslab.stanford.edu/papers/FrankeDegen_submitted.pdf (дата обращения: 04.06.2025).

14. Wrestling Mamba: Exploring Early Fine-Tuning Dynamics on Mamba and Transformer Architectures [Электронный ресурс]: сайт Stanford. URL:https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/final-projects/DanielGuoLucasEmmanuelBrennanAlmaraz.pdf (дата обращения: 04.06.2025).

15. Hardware-aware Algorithms for Sequence Modelin [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture18-deployment-and-efficiency.pdf (дата обращения: 04.06.2025).

16. AI CanLook UpStackOverflow too: Retrieval-Augmented Code Generation [Электронный ресурс]: сайт Stanford. URL: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1244/slides/cs224n-2024-lecture18-deployment-and-efficiency.pdf (дата обращение: 04.06.2025).

17. IBM Watson [Электронный ресурс]: сайт IBM Watson. URL: https://www.ibm.com/watson (дата обращение: 04.06.2025).

18. Azure AI services [Электронный ресурс]: Microsoft Learn Challenge. URL: https://learn.microsoft.com/en-us/azure/ai-services/ (дата обращение: 04.06.2025).

19. Google Cloud AI Platform [Электронный ресурс]: сайт Google Cloud. URL: https://cloud.google.com/ai (дата обращения: 04.06.2025).

20. Amazon Lex [Электронный ресурс]: сайт Amazon Web Services. URL: https://aws.amazon.com/lex/ (дата обращение: 04.06.2025).

21. GPT-4 [Электронный ресурс]: сайт OpenAI. URL: https://openai.com/index/gpt-4/ (дата обращение: 04.06.2025).

22. Hugging Face Chat [Электронный ресурс]: сайт Hugging Face. URL: https://huggingface.co/ (дата обращение: 04.06.2025).

23. DeepSeek [Электронный ресурс]: сайт DeepSeek. URL: https://www.deepseek.com/ (дата обращение: 04.06.2025).

24. GigaChat [Электронный ресурс]: сайт GigaChat. URL: https://giga.chat/ (дата обращение: 04.06.2025).

25. Юридические документы GigaChat [Электронный ресурс]: сайт GigaChat. URL: https://developers.sber.ru/docs/ru/policies/gigachat-agreement/general (дата обращение: 04.06.2025).

26. Jupyter Notebook [Электронный ресурс]: сайт Jupyter. URL: https://jupyter.org/ (дата обращение: 04.06.2025).

27. VS Code [Электронный ресурс]: сайт Visual Studio Code. URL: https://code.visualstudio.com/ (дата обращение: 04.06.2025).

28. Streamlit [Электронный ресурс]: сайт Streamlit. URL: https://streamlit.io/ (дата обращение: 04.06.2025).

29. Python [Электронный ресурс]: сайт Python. URL: https://www.python.org/ (дата обращение: 04.06.2025).

30. UV [Электронный ресурс]: сайт Astral. URL: https://docs.astral.sh/uv/ (дата обращение: 04.06.2025).

Приложение А
(обязательное)
Диаграмма прецедентов

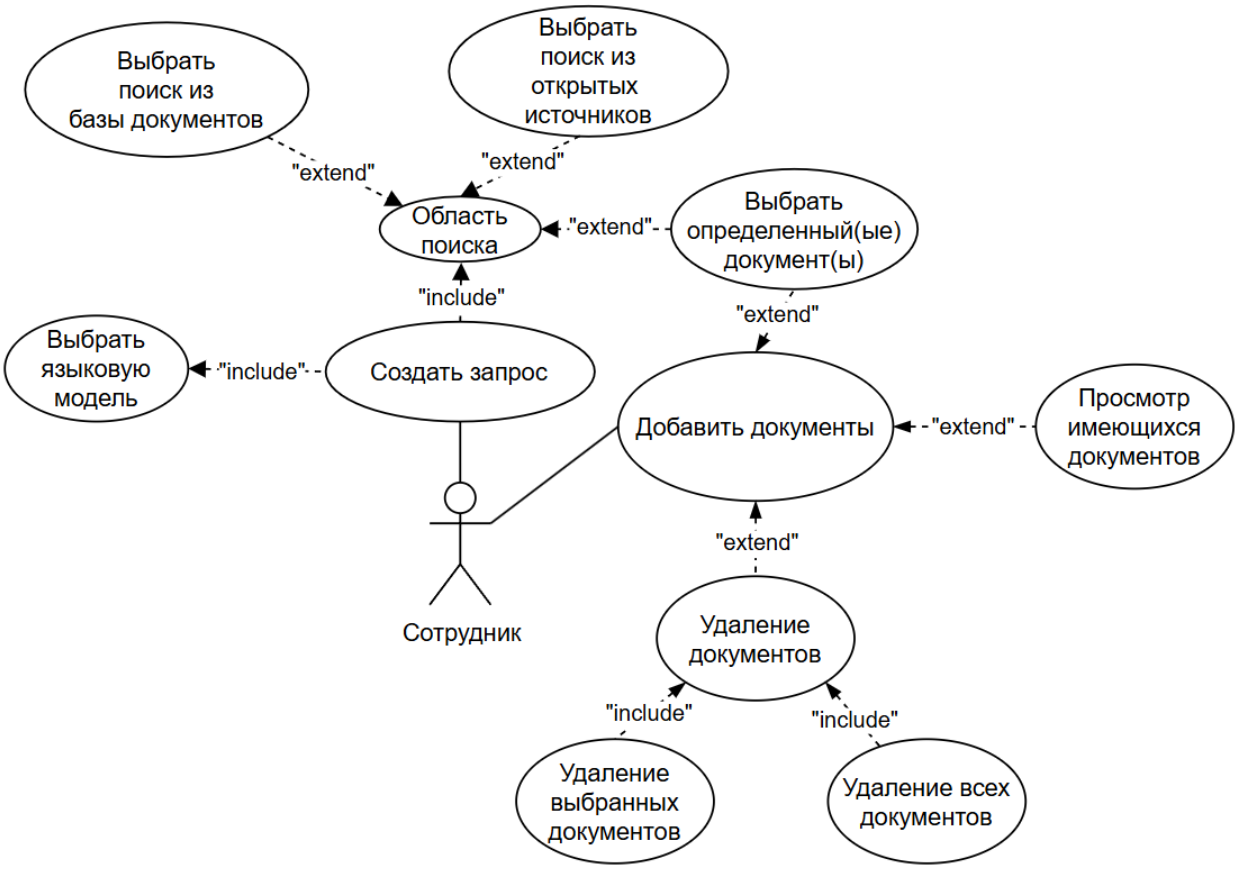

Рисунок А.1 – Диаграмма прецедентов

Приложение Б
(обязательное)
Диаграмма последовательности

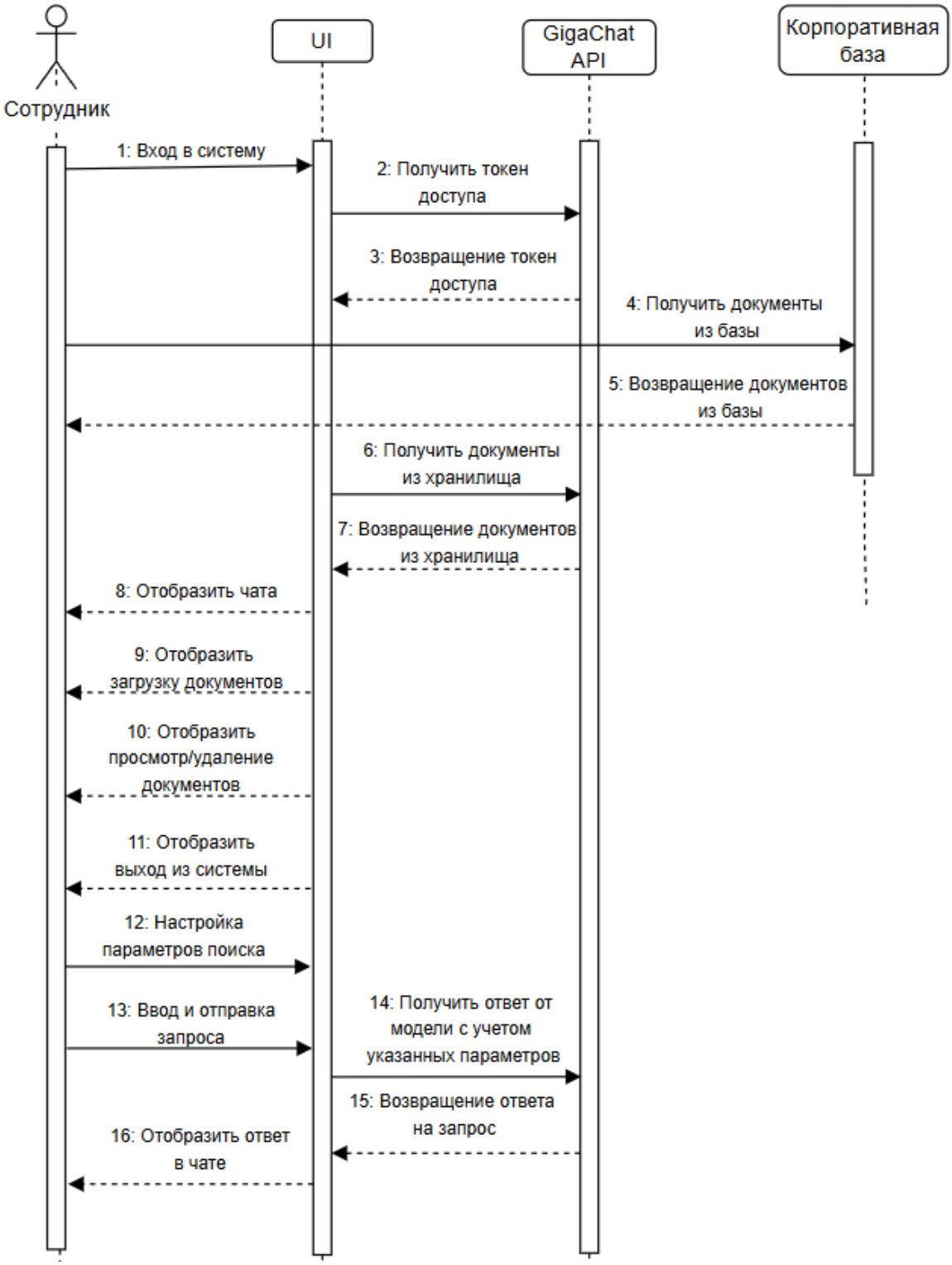

Рисунок Б.1 – Диаграмма последовательности этапа получения ответа на запрос

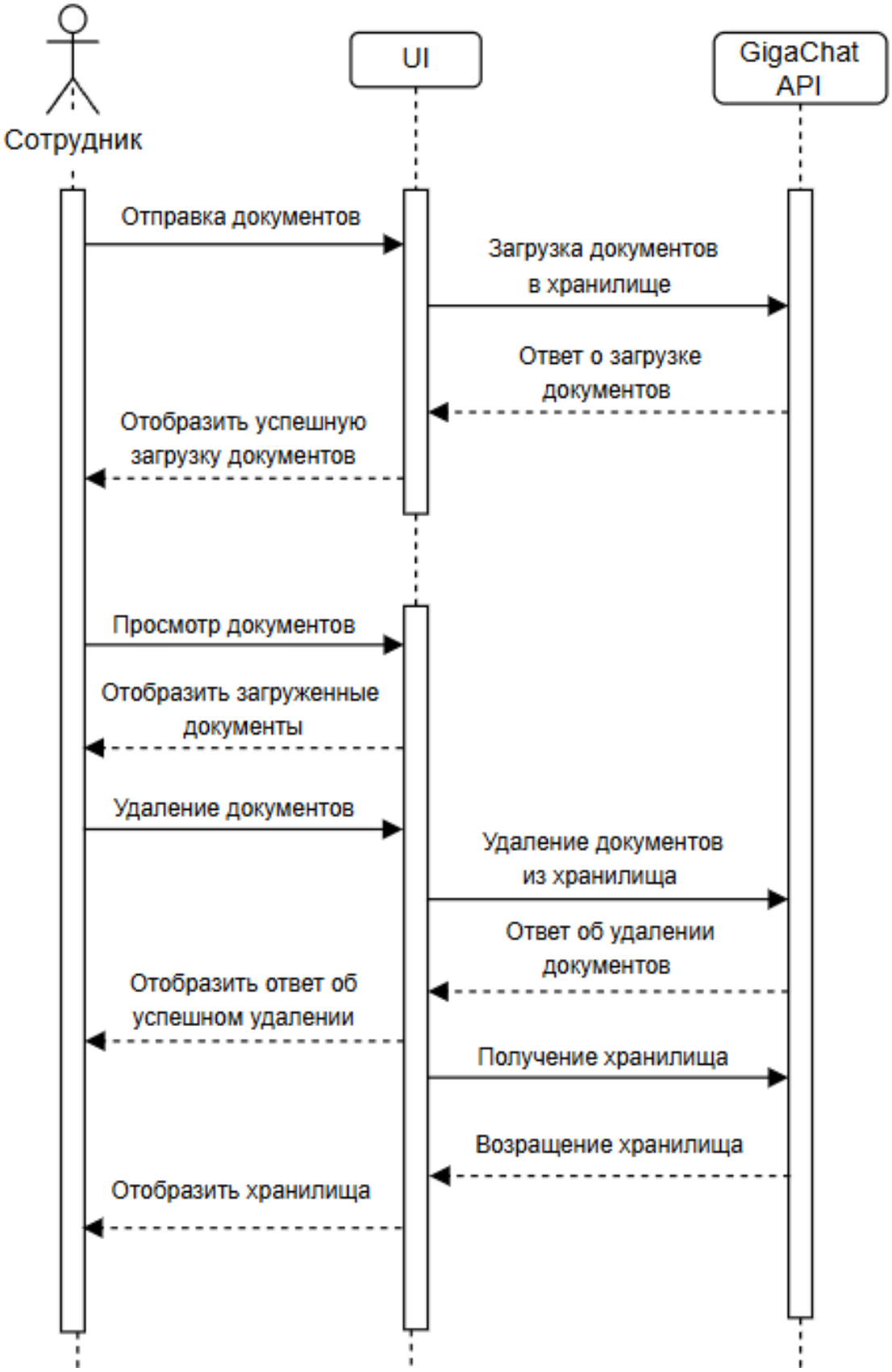

Рисунок Б.2 – Диаграмма последовательности этапа загрузки, просмотра, удаление документов из хранилища

Приложение В
(обязательное)
Макеты интерфейса

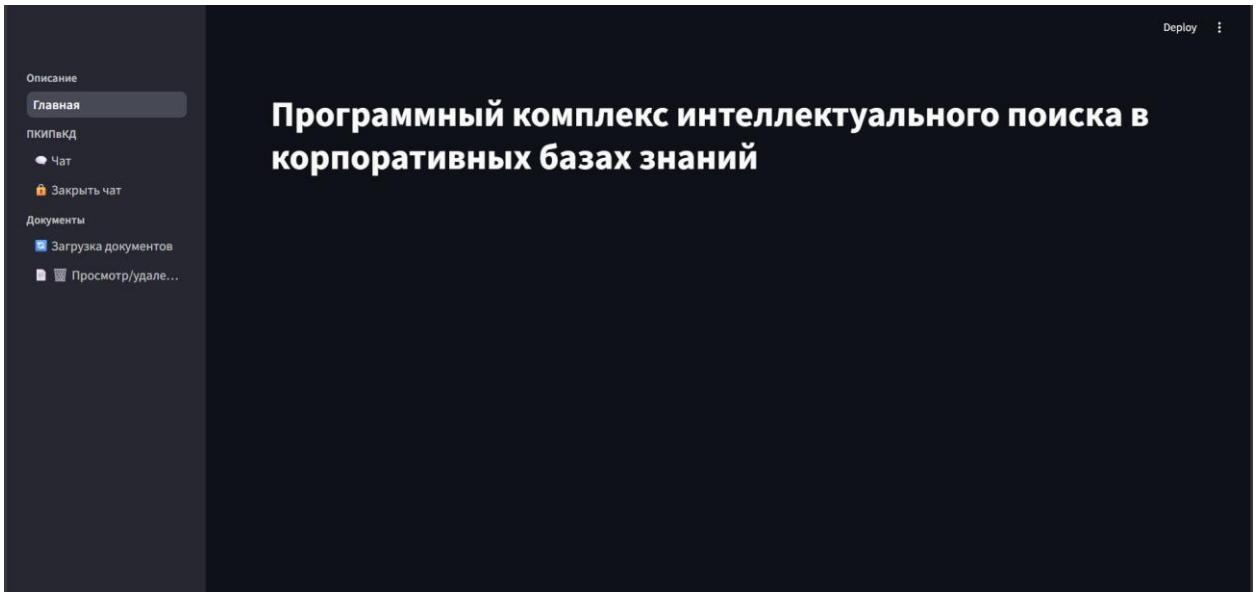

Рисунок В.1 – Макет главной страницы системы с описанием взаимодействия

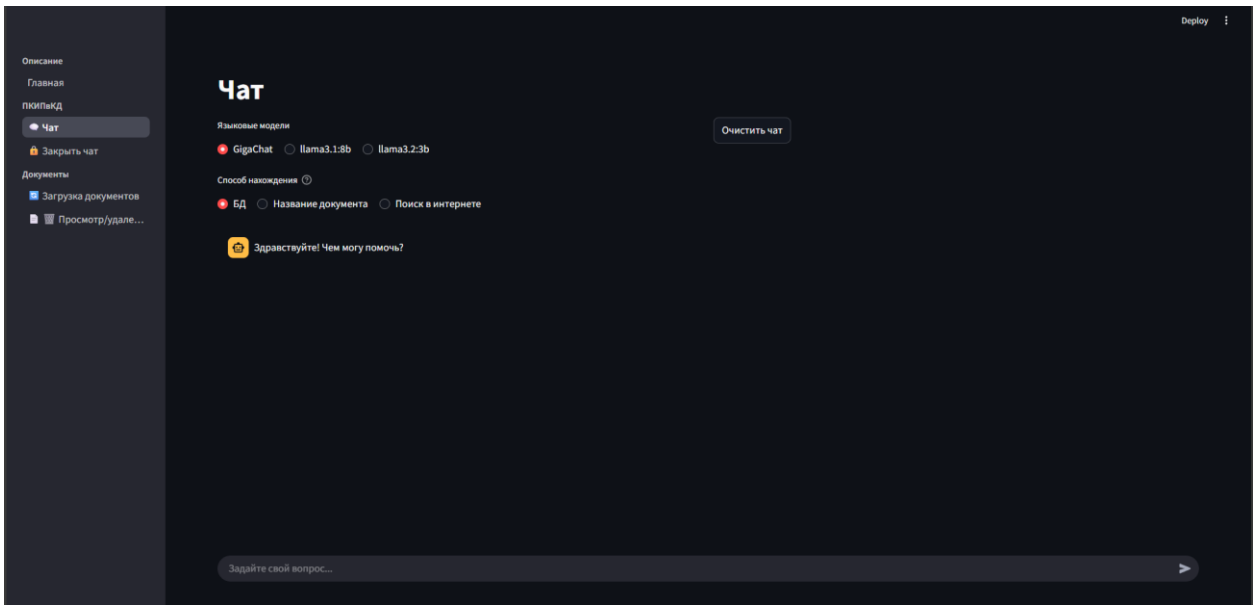

Рисунок В.2 – Макет чата системы

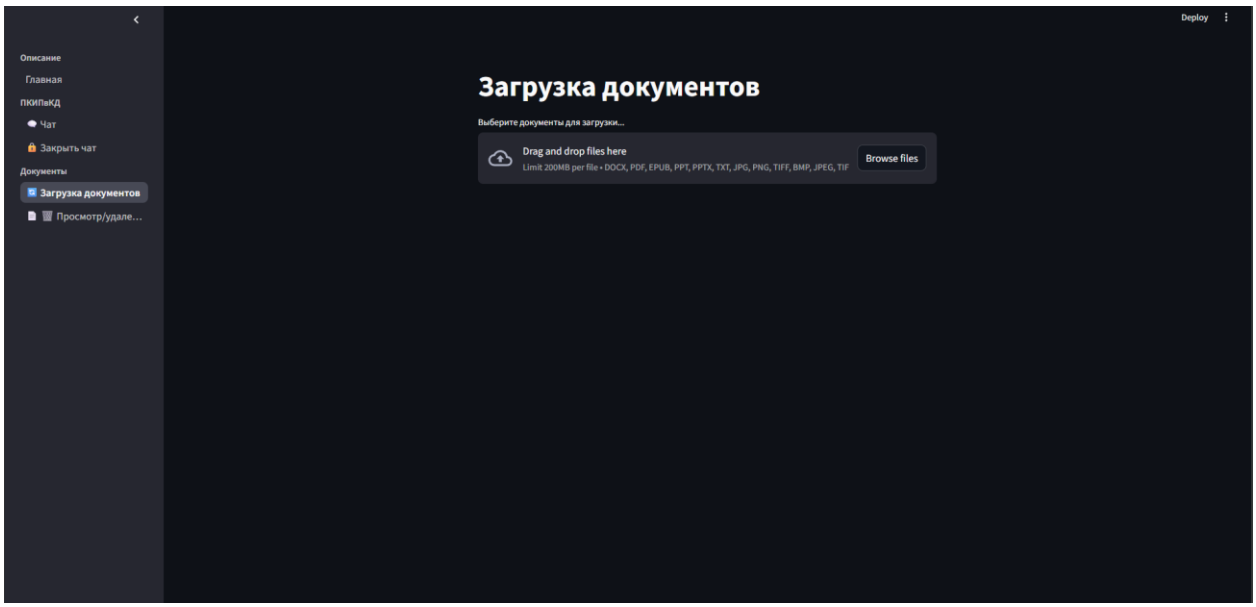

Рисунок В.3 –Макет загрузки документов с предоставление справочной информации

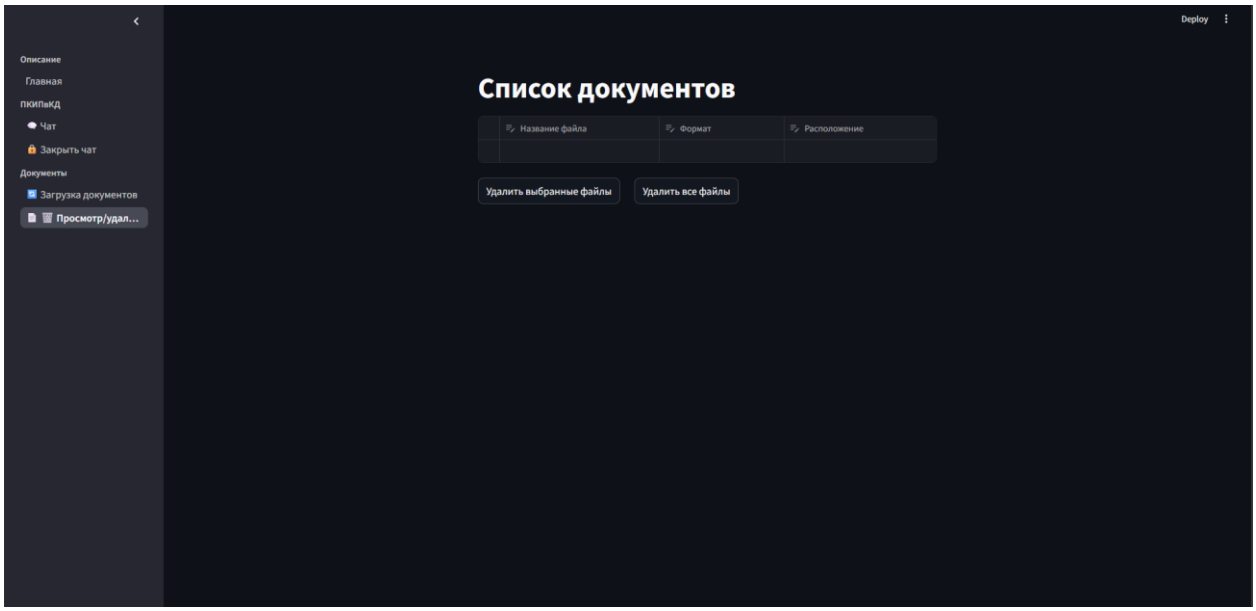

Рисунок В.4 –Макет просмотра/удаления документов из хранилища

Приложение Г
(обязательное)
Диаграмма пакетов

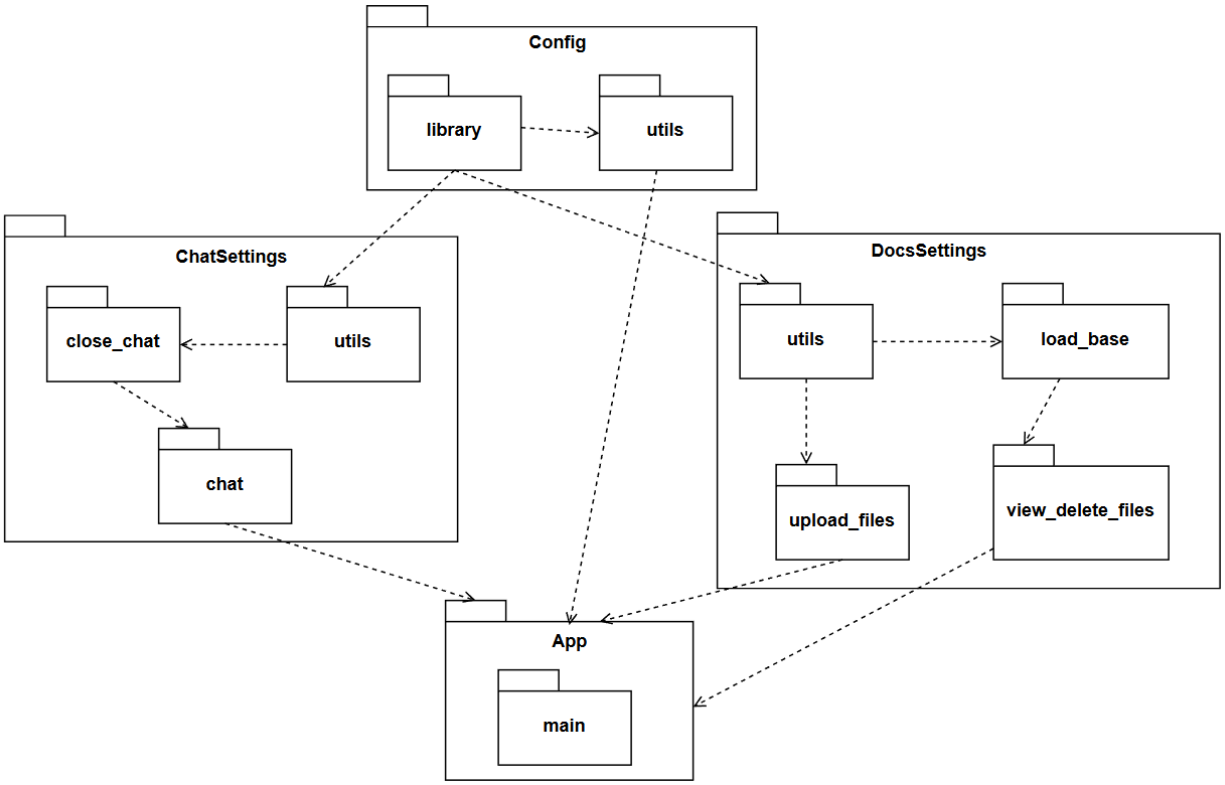

Рисунок Г.1 – Диаграмма пакетов

Приложение Д
(обязательное)
Конфигурация проекта

Листинг Д.1 – Конфигурация проекта venv в файле pyproject.toml

```
[project]
name = "venv"
version = "0.1.0"
description = ""
readme = "README.md"
requires-python = ">=3.12"
dependencies = [
    "langchain==0.3.25",
    "langchain-community==0.3.24",
    "langchain-gigachat==0.3.10",
    "langchain-ollama==0.3.3",
    "langchain-text-splitters==0.3.8",
    "pandas==2.2.3",
    "psutil==7.0.0",
    "psycopg2==2.9.10",
    "pypdf==5.5.0",
    "python-docx==1.1.2",
    "python-pptx==1.0.2",
    "requests==2.32.3",
    "streamlit==1.45.1",
    "uv==0.7.8",
]
```

Листинг Д.2 – Конфигурация secrets.toml

```
CLIENT_ID="ключ_доступа"
SECRET="ключ_доступа"
AUTH_KEY="ключ_доступа"
```

[система_бд]

host="адрес"

port="номер_порта"

dbname="имя_базы_данных"

user="пользователь"

password="пароль"

[система_бдlink]

link="система_бд://пользователь:пароль@адрес/имя_базы_данных"

Приложение Е
(обязательное)
Реализация программного комплекса

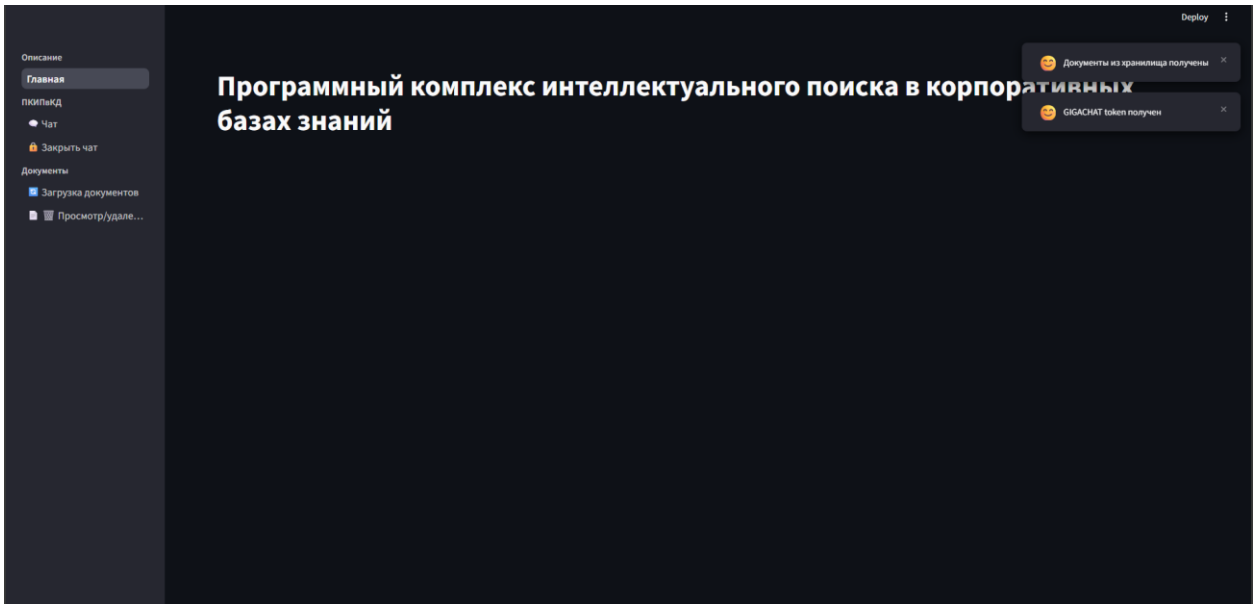

Рисунок Е.1 – Инициализация необходимых данных для работы системы

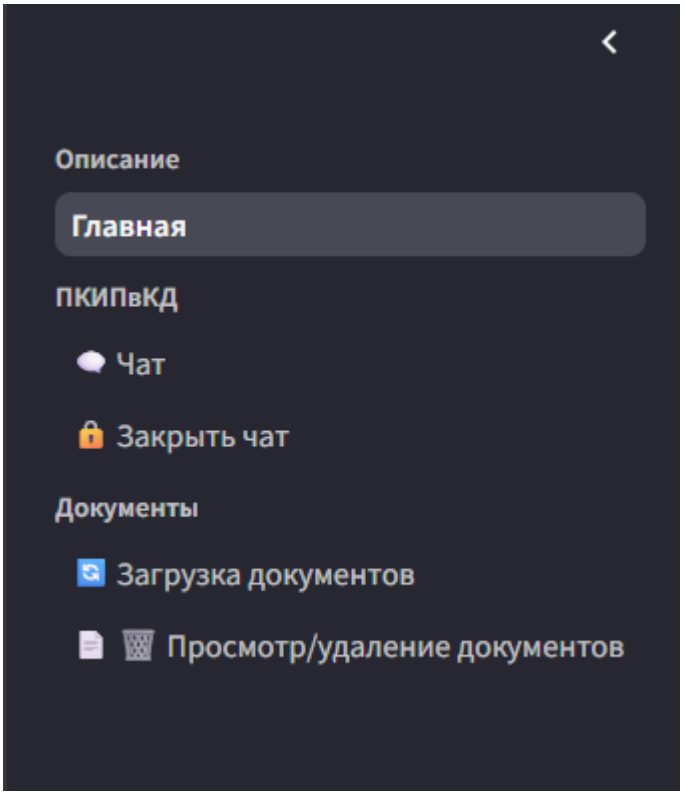

Рисунок Е.2 – Панель перемещения по страницам системы

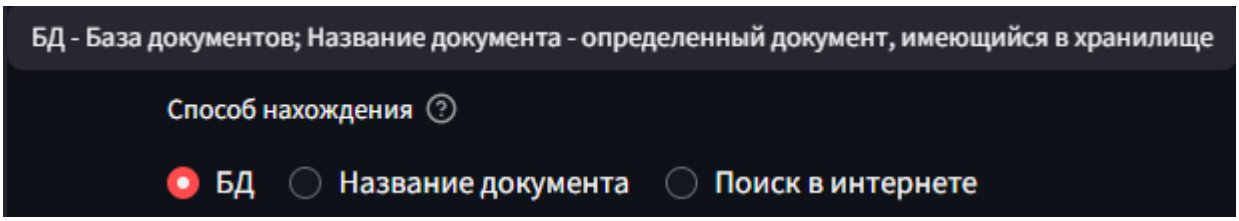

Рисунок Е.3 – Всплывающая подсказка при наведении на параметр

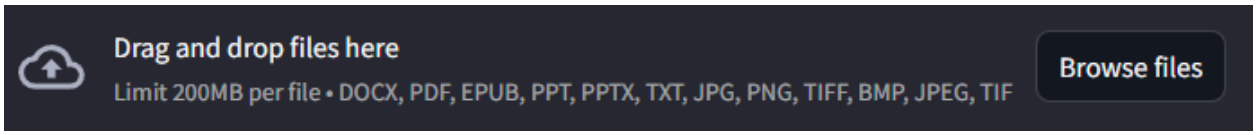

Рисунок Е.4 – Подсказка на самом виджете

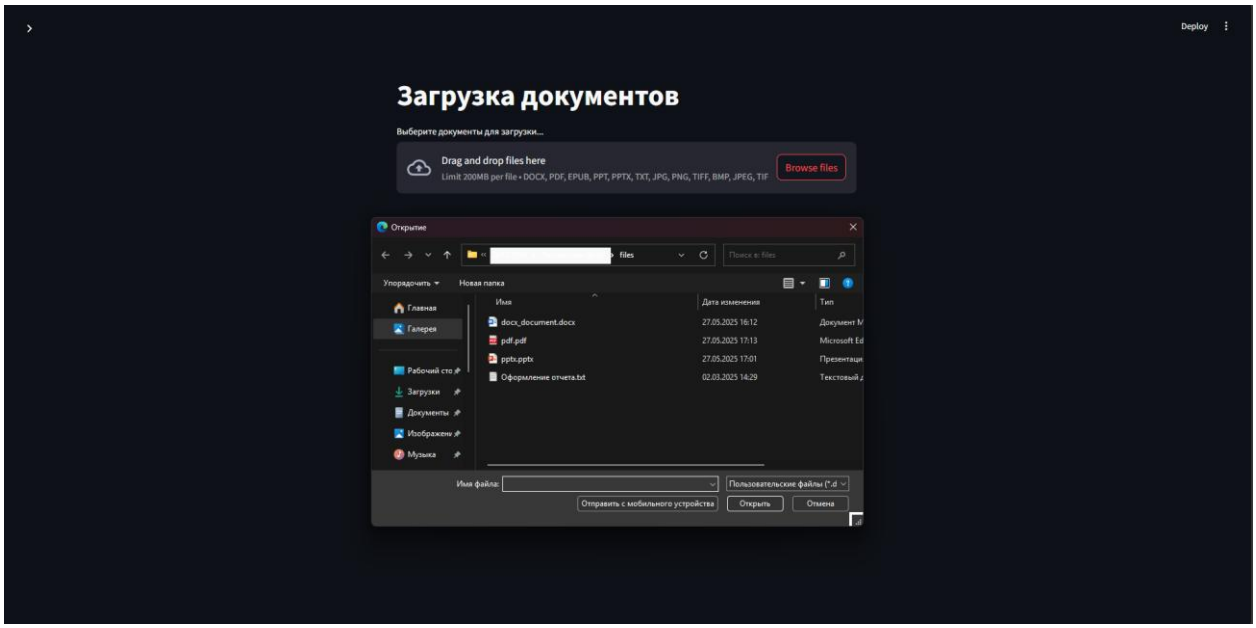

Рисунок Е.5 – Выбор документов при помощи вызова интерфейса из операционной системы

Приложение Ж
(обязательное)
Тестирование программного комплекса

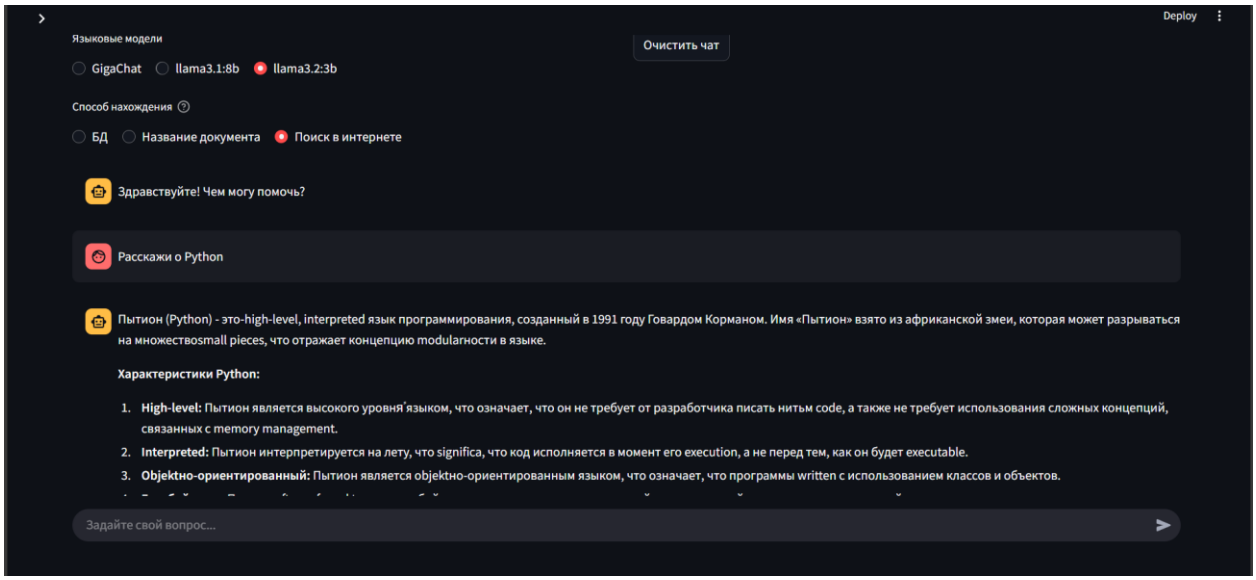

Рисунок Ж.1 – Ответ llama3.2:3b из открытых источников

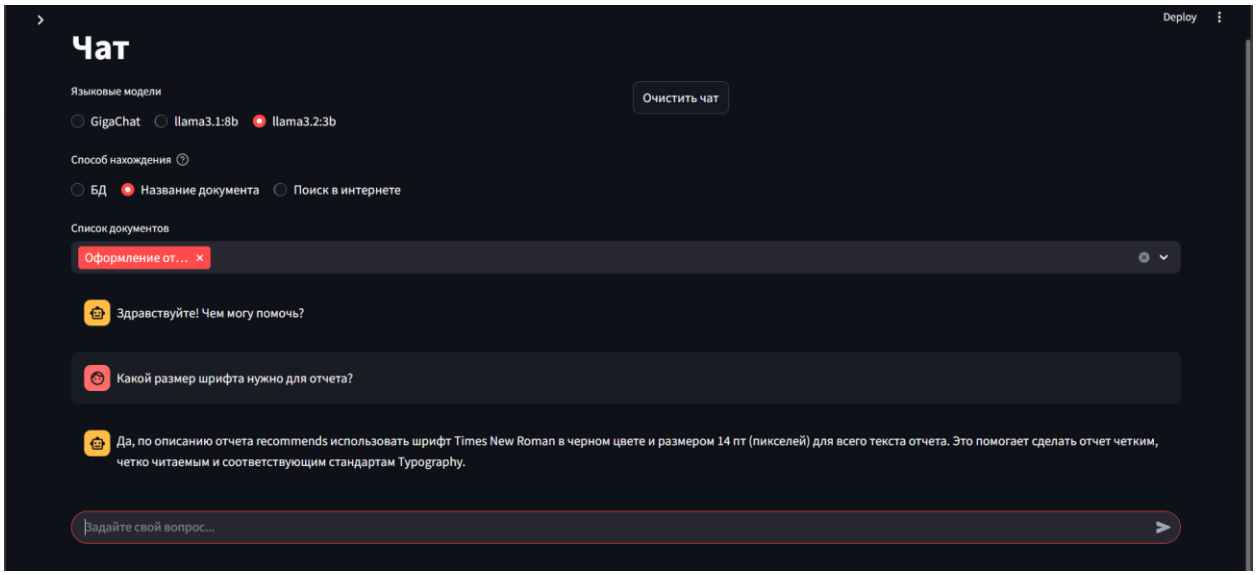

Рисунок Ж.2 – Ответ llama3.2:3b из выбранного документа

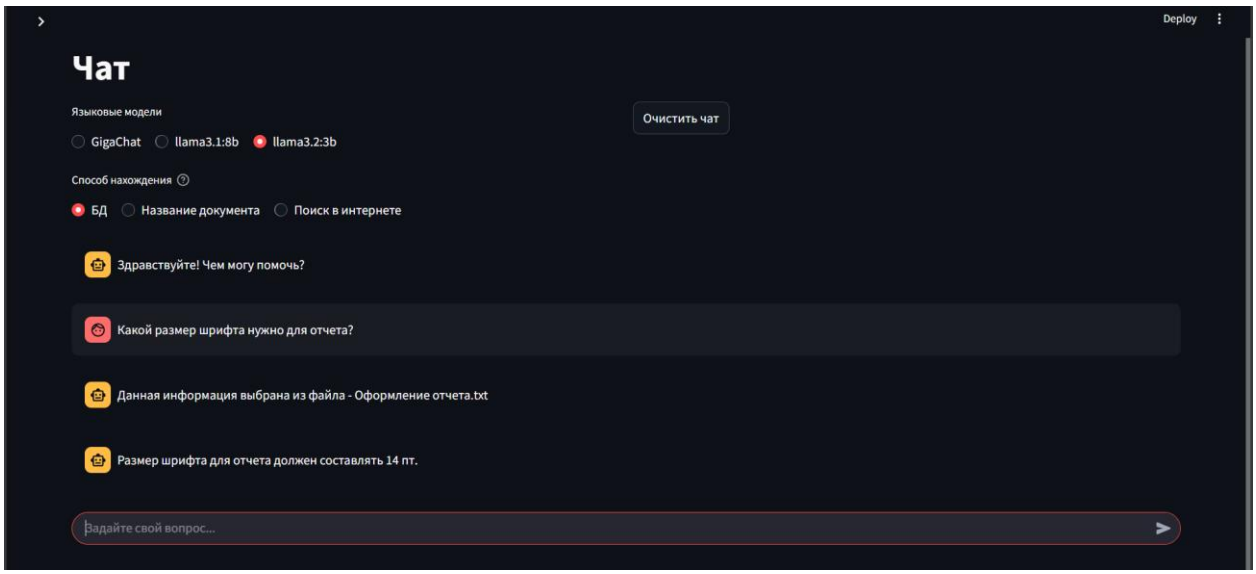

Рисунок Ж.3 – Ответ llama3.2:3b из корпоративной базы

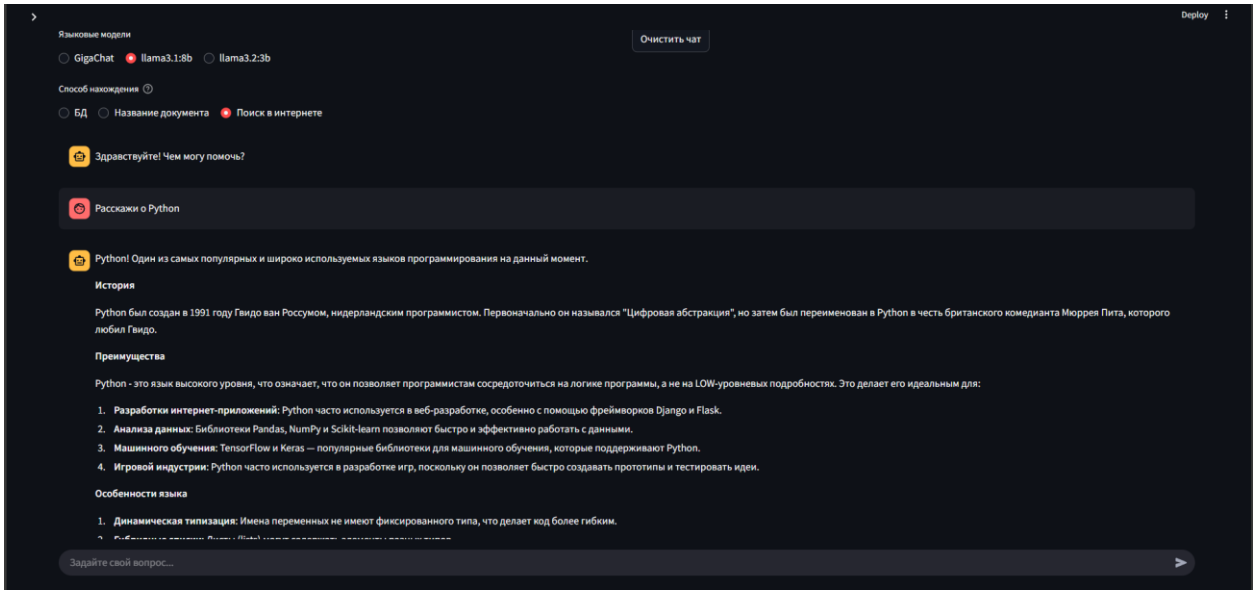

Рисунок Ж.4 – Ответ llama3.1:8b из открытых источников

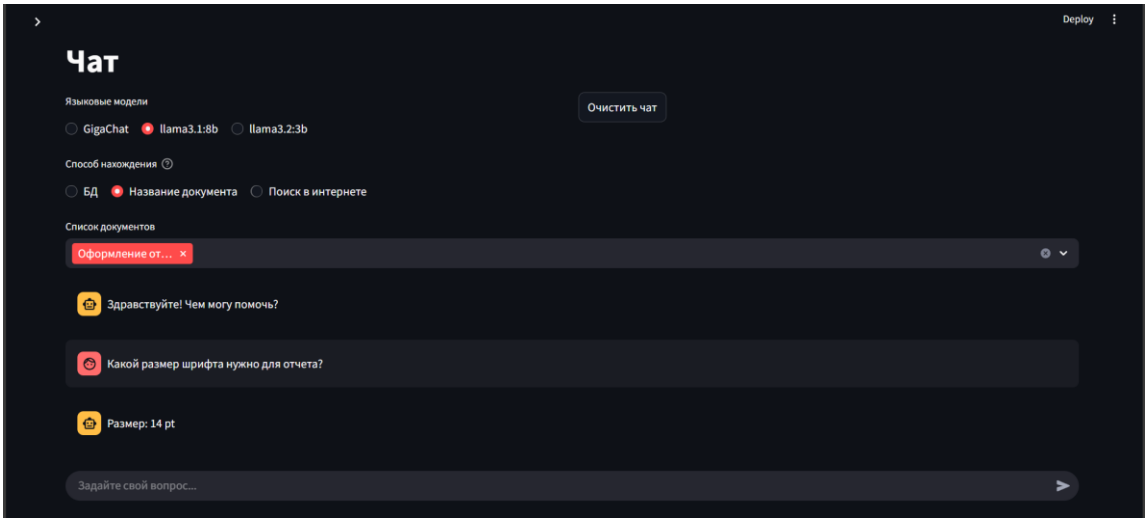

Рисунок Ж.5 – Ответ llama3.1:8b из выбранного документа

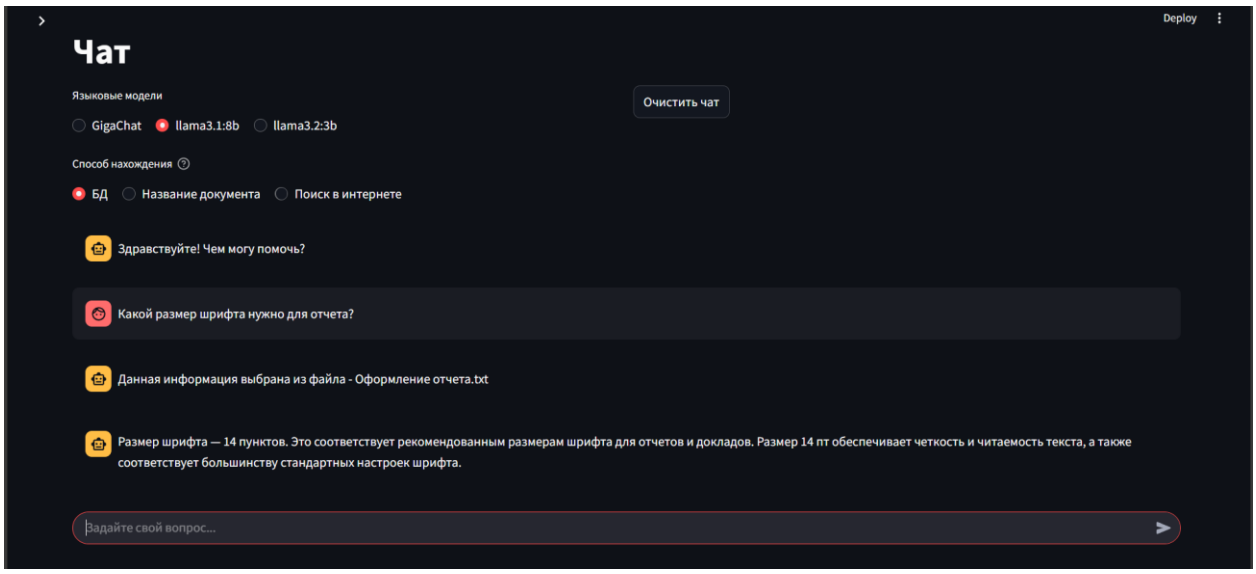

Рисунок Ж.6 – Ответ llama3.1:8b из корпоративной базы

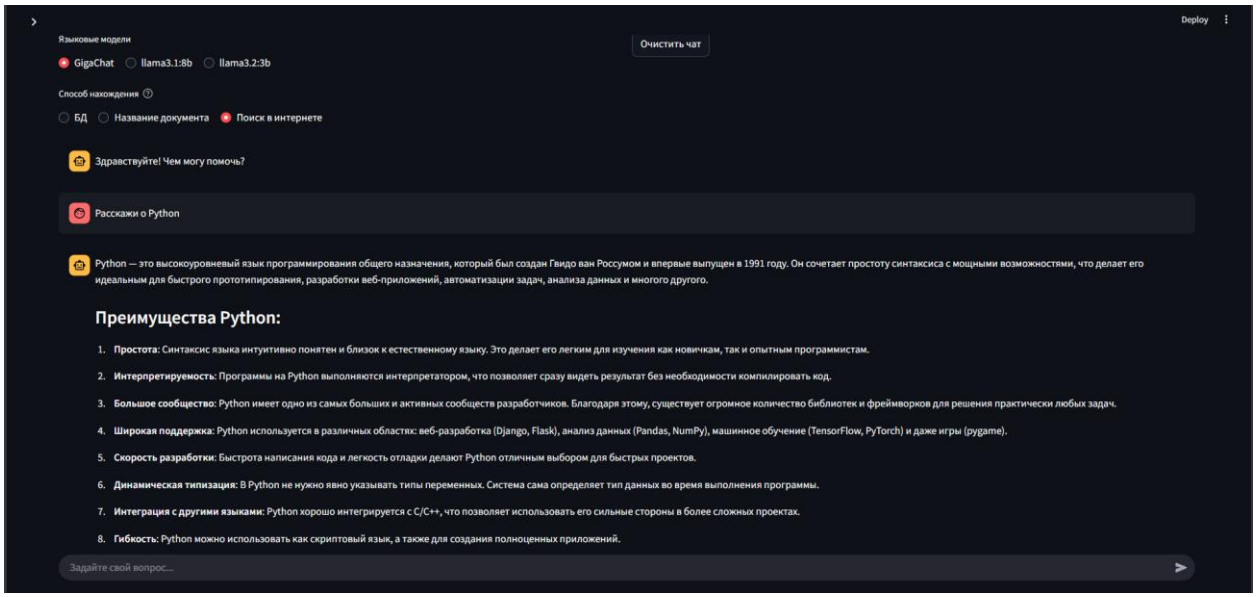

Рисунок Ж.7 – Ответ GigaChat из открытых источников

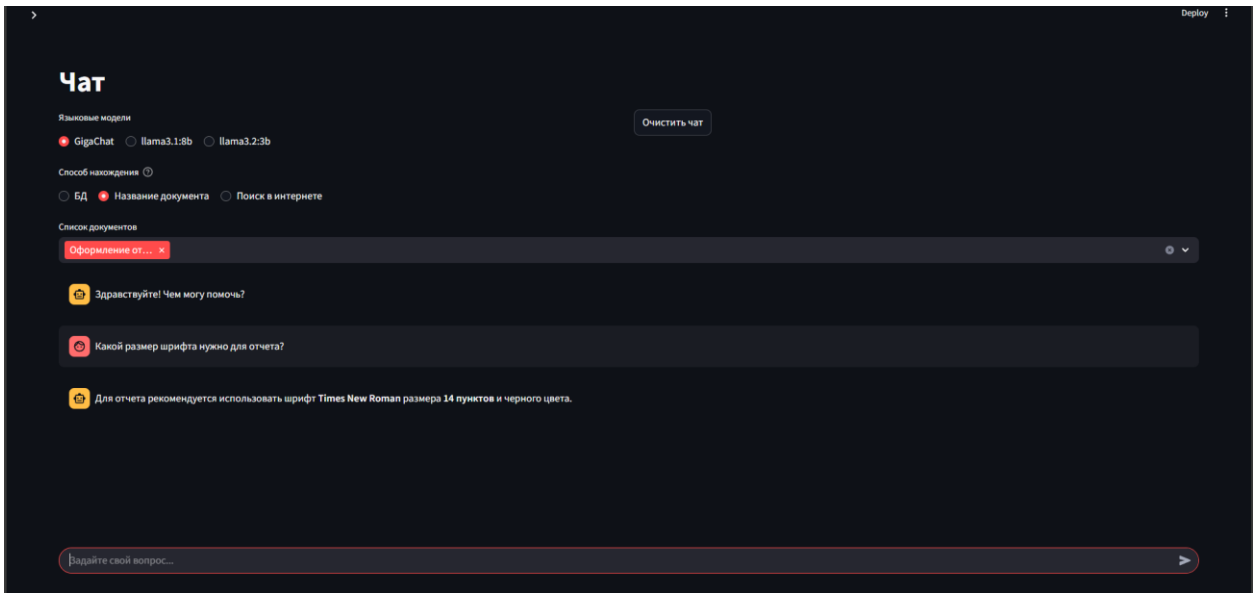

Рисунок Ж.8 – Ответ GigaChat из выбранного документа

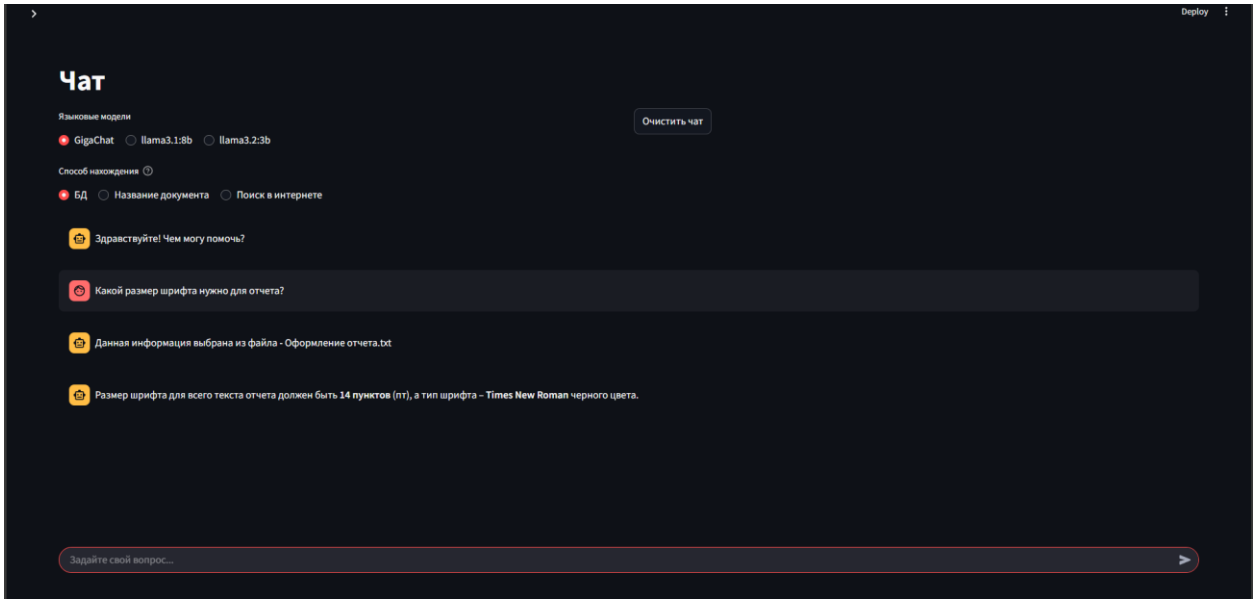

Рисунок Ж.9 – Ответ GigaChat из корпоративной базы

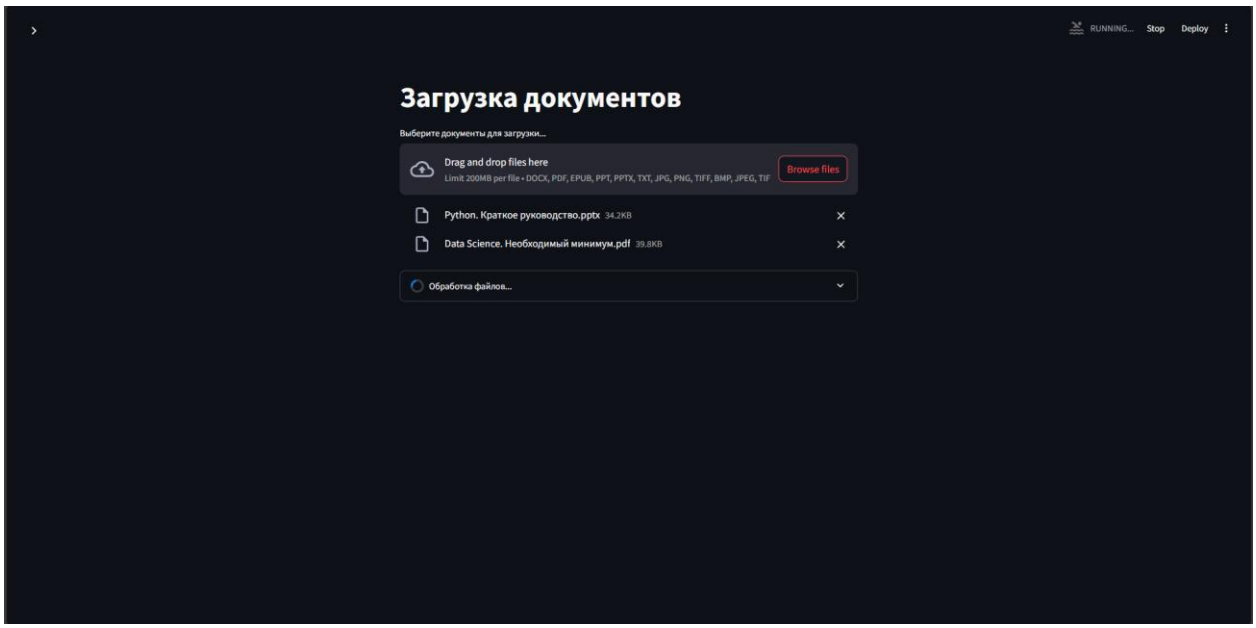

Рисунок Ж.10 – Загрузка документов в хранилище

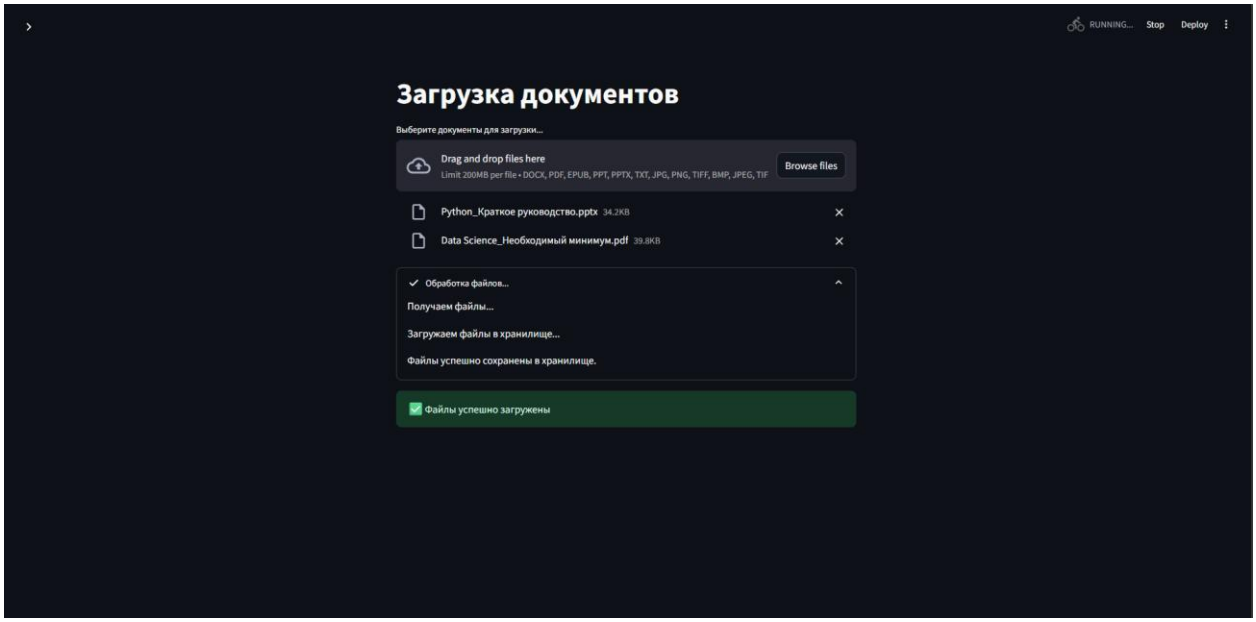

Рисунок Ж.11 – Успешная загрузка документов в хранилище

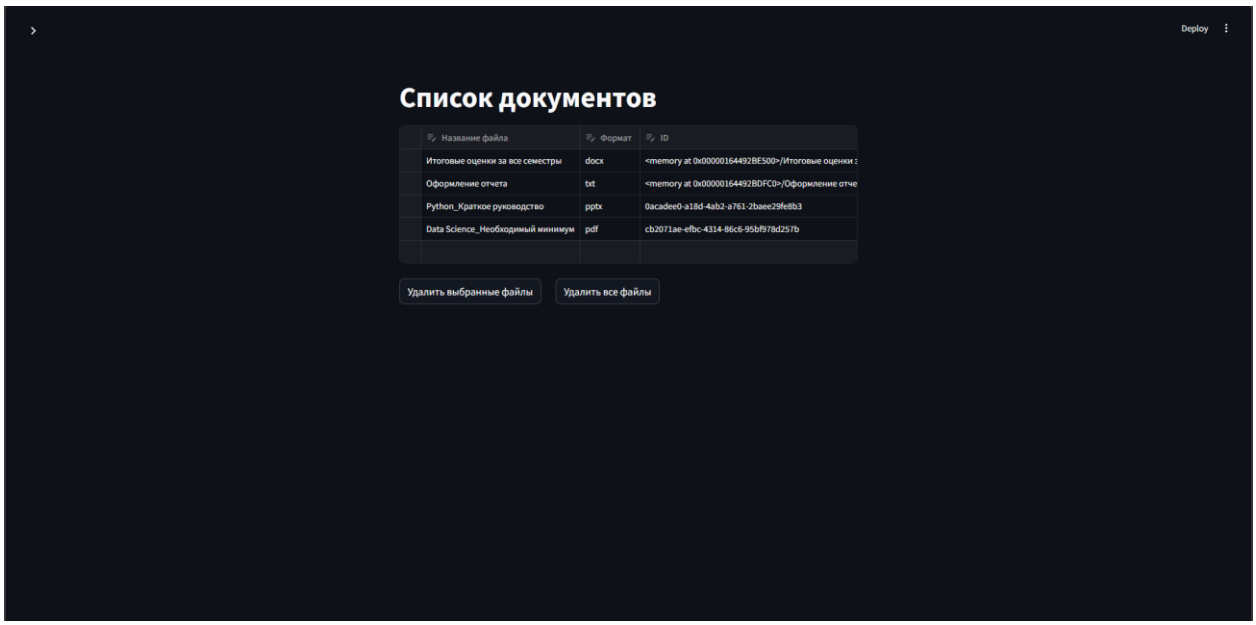

Рисунок Ж.12 – Просмотр текущих документов в хранилище и корпоративной базы

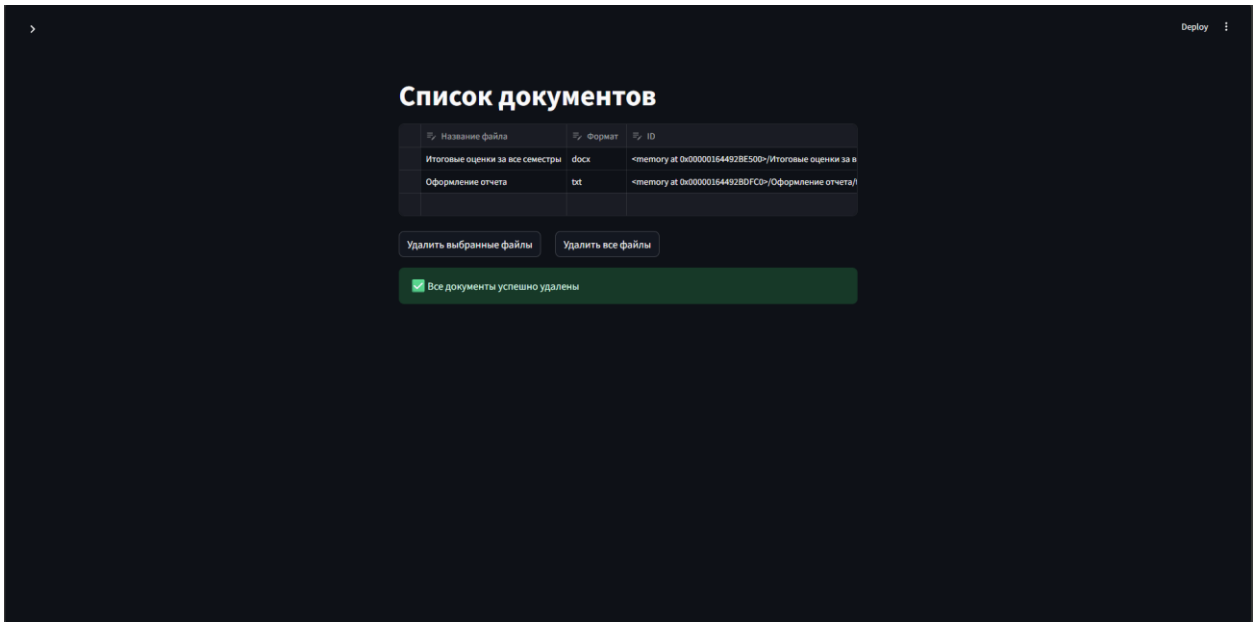

Рисунок Ж.13 – Успешное удаление документов из личного хранилища