МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
**«Южно-Уральский государственный университет
(национальный исследовательский университет)»**
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

ДОПУСТИТЬ К ЗАЩИТЕ

Заведующий кафедрой, д.ф.-м.н., профессор

_____ Л.Б. Соколинский

«___»_____ 2024 г.

Разработка чат-бота для ответов на вопросы по продукту TIP

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
ЮУрГУ – 09.03.04.2024.308-367.ВКР

Научный руководитель,
доцент кафедры СП, к.ф.-м.н.
_____ С.А. Иванов

Автор работы,
студент группы КЭ-404
_____ Д.В. Тропин

Ученый секретарь
(нормоконтролер)
_____ И.Д. Володченко
«___»_____ 2024 г.

Челябинск, 2024 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Южно-Уральский государственный университет
(национальный исследовательский университет)»
Высшая школа электроники и компьютерных наук
Кафедра системного программирования

УТВЕРЖДАЮ
Зав. кафедрой СП
_____ Л.Б. Соколинский
29.01.2024 г.

ЗАДАНИЕ
на выполнение выпускной квалификационной работы бакалавра
студенту группы КЭ-404
Тропину Дмитрию Вячеславовичу,
обучающемуся по направлению
09.03.04 «Программная инженерия»

1. **Тема работы** (утверждена приказом ректора от 22.04.2024 г. № 764-13/12)

Разработка чат-бота для ответов на вопросы по продукту TIP.

2. **Срок сдачи студентом законченной работы:** 03.06.2024 г.
3. **Исходные данные к работе**

3.1. Шумилина М.А., Коробко А.В. Разработка чат-бота на языке программирования Python в мессенджере «Telegram». // Научные известия, 2022. – №28. – С. 47–54.

3.2. Жеребцова Ю.А., Чижик А.В. Сравнение моделей векторного представления текстов в задаче создания чат-бота. // Вестник НГУ, Серия: Лингвистика и межкультурная коммуникация, 2020. – №3. – С. 16–34.

3.3. Puri R., Spring R., Patwary M., Shoeybi M., Catanzaro B. Training Question Answering Models From Synthetic Data. // EMNLP, 2020. – С. 5811–5826.

4. **Перечень подлежащих разработке вопросов**

4.1. Провести анализ предметной области и обзор существующих решений.

4.2. Спроектировать и реализовать чат-бот.

4.3. Провести ряд экспериментов для подбора оптимальных параметров системы.

4.4. Провести тестирование разработанного чат-бота.

5. Дата выдачи задания: 29.01.2024 г.

Научный руководитель,
доцент кафедры СП, к.ф.-м.н. С.А. Иванов

Задание принял к исполнению Д.В. Тропин

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Актуальность

В настоящий момент одно из самых быстроразвивающихся направлений в сфере информационных технологий является создание систем способных отвечать на различные вопросы пользователей и поддерживать человеко-машинный диалог.

Появление технологий искусственного интеллекта (ИИ) и обработки естественного языка (NLP) произвело революцию в системах обслуживания клиентов и поддержки. Чат-бот – это одна из разновидностей данных систем, представляющая собой виртуального помощника с ограниченным функционалом. Диалог между пользователем и чат-ботом осуществляется посредством текстовых сообщений [1].

Широкое применение чат-боты получили в различных сферах общества, особенно часто они внедряются в образование, государственные электронные порталы и коммерческие бизнес решения, на это есть ряд причин.

1. Чат-боты способны значительно повысить удовлетворенность клиентов, так как пользователь может моментально получить актуальную информацию.
2. Автоматизация ответов на вопросы клиентов, положительно влияет на оптимизацию бизнес-процессов в компании за счет освобождения времени специалистов для решения более сложных задач.
3. Собираемые ботами данные о часто задаваемых вопросах и запросах пользователей могут быть использованы для проведения анализа и улучшения качества продуктов и услуг компании на основе обратной связи от клиентов.

Таким образом, разработка чат-бота для продукта R-Vision TIP является актуальной задачей, способствующей повышению конкурентоспособности организаций и оптимизации бизнес-процессов.

Постановка задачи

Целью выпускной квалификационной работы является разработка чат-бота для ответов на вопросы по продукту TIP.

Для достижения поставленной цели необходимо решить следующие задачи:

1) провести анализ предметной области и произвести обзор существующих решений;
2) спроектировать и реализовать чат-бот;
3) провести ряд экспериментов для подбора параметров системы;
4) провести тестирование разработанного чат-бота.

Структура и содержание работы

Работа состоит из введения, четырех разделов, заключения и списка литературы. Объем работы составляет 55 страниц, объем списка литературы – 30 наименований.

В первой главе описывается анализ предметной области и обзор существующих решений.

Во второй главе приведены функциональные и нефункциональные требования к системе, диаграмма вариантов использования, а также представлено общее описание архитектуры системы.

В третьей главе описан процесс разработки системы, представлены листинги исходных кодов, а также результаты экспериментов по подбору параметров системы.

Четвертая глава описывает процесс тестирования системы, включая модульное и функциональное тестирование.

В приложении А содержится спецификация вариантов использования.

В приложении Б содержится листинги исходных кодов системы.

В приложении В приведен листинг разработанных unit-тестов.

В приложении Г приведен процесс регистрации чат-бота.

В приложении Д представлена диаграмма деятельности.

1. АНАЛИЗ ПРЕДМЕТНОЙ ОБЛАСТИ И ОБЗОР СУЩЕСТВУЮЩИХ РЕШЕНИЙ

1.1. Описание предметной области

Целью данной работы является разработка чат-бота для ответов на вопросы по продукту TIP. Чат-бот – это разговорно-диалоговая система, с которой пользователь взаимодействует на естественном языке [2].

Существует три основные разновидности данных систем с точки зрения бизнес логики.

1. Разговорные чат-боты – созданы для общения, диалога с человеком, обычно не имеют конкретной цели.

2. Чат-боты ассистенты – имеют конкретные заранее определенные цели. Из пользовательских ответов извлекаются данные для достижения этих целей. Обычно служат ассистентами для заполнения Web-форм.

3. Вопросно-ответные боты (Q&A, questions and answers) – чат-боты разрабатываемые, чтобы давать ответы по принципу один вопрос – один ответ. Широко применяются в бизнес решениях в качестве FAQ (frequently asked questions) раздела.

Что касается технической реализации чат-боты имеют следующую классификацию.

1. Основанные на бизнес-правилах – такие системы имеют дерево диалога, то есть диалог с пользователем ведется по определенному алгоритму.

2. Основанные на алгоритмах машинного обучения – такие чат боты реализуются на основе моделей обработки естественного языка (NLP – natural language processing).

3. Гибридные системы – являются комбинацией двух предыдущих типов [3].

Так как разрабатываемый чат-бот имеет определенно цель, а именно ответы на вопросы по продукту R-Vision TIP, следовательно, данная система с точки зрения бизнес логики является вопросно-ответным ботом.

Данное решение не подразумевает полноценного диалога с пользователем, а это значит, для решения поставленной задачи наиболее целесообразно реализовывать чат-бота на основе алгоритмов машинного обучения.

Имеется несколько основных концепций в отношении моделей обработки естественного языка для вопросно-ответных систем.

Использование векторного поиска по базе данных готовых ответов – данных подход является наиболее простым с точки зрения реализации и сопровождения, однако ответы получаются довольно простыми, однотипными и неполными, так как очень сложно покрыть ответами все возможные вопросы. В данном случае модели машинного обучения используются для преобразования текста вопроса в вектор (эмбеддинг), в дальнейшем производится поиск по векторной базе данных с помощью различных алгоритмов нахождения ближайших векторов.

Retriever and Reader – данные системы ведут поиск по базе знаний, извлекают необходимые данные и выстраивают предположение для построения ответа. Модуль retriever предназначен для извлечения объектов, которые могут принадлежать входному тексту, на основании базы знаний, а целью модуля reader является дальнейшее сокращение набора данных кандидатов и прогнозирование конкретной информации о местоположении в интервале окончательно предсказанных объектов в тексте [4]. Схема данной архитектуры приставлена на рисунке 1.

Генеративные Q&A модели – данные модели обучаются на вопросах и ответах на них, что делает их строго направленными для определенной области, а для дополнения системы новыми знаниями каждый раз требуется дообучение. Однако при использовании данного подхода ответы получаются наиболее приближенными к ответам, которые бы составил специалист.

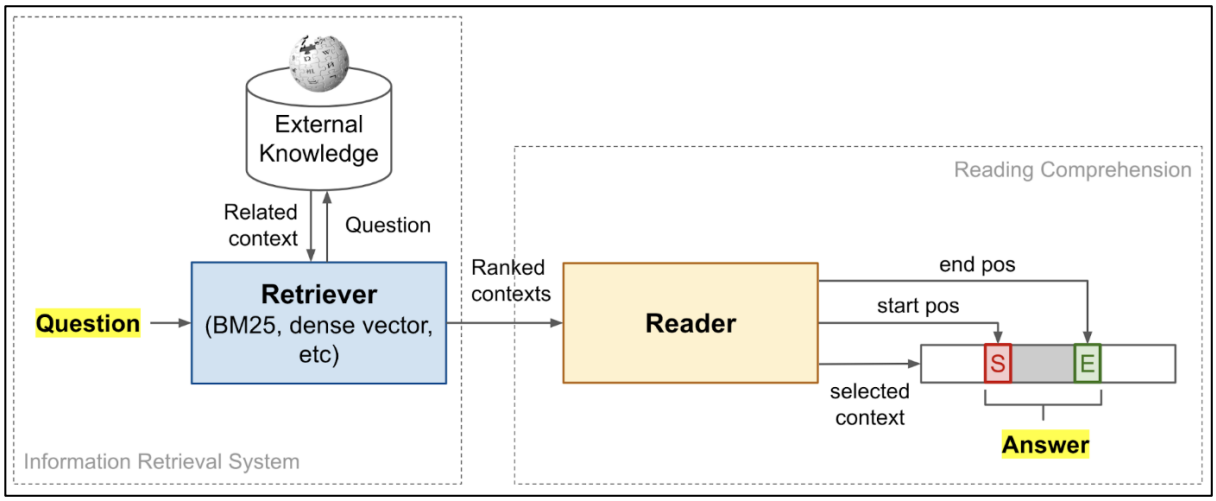


Рисунок 1 – Retriever-Reader архитектура

Учитывая область применения, а именно ответы на вопросы клиентов, можно сделать вывод, что наиболее подходящим путем является обучение генеративной Q&A модели на ответах и вопросах по документации продукта, а также взятых из корпоративного чата.

В основе выбранного подхода лежат предобученные модели, построенные на архитектуре Transformer, к ним относятся такие модели как GPT, BART, T5, BERT, RoBERTa и XLNet. Все перечисленные модели были предобучены как языковые модели – это означает, что они были обучены на больших объемах необработанного текста под самоконтролем. Обучение с самоконтролем – это тип обучения, при котором цель автоматически вычисляется на основе входных данных модели [5].

В первые данная архитектура была представлена в декабре 2017 года в статье «Attention is All You Need», разработчики нейронной архитектуры transformer были заинтересованы в поиске архитектуры, которая могла бы работать для моделирования последовательностей [6]. На данный момента Transformer является главной концепцией построения архитектуры нейронных сетей глубокого обучения для задач обработки естественного языка (NLP).

Итак, Transformer – это архитектура глубокой нейронной сети, которая может обрабатывать последовательные данные. Модель Transformer

состоит из блоков, каждый из которых включает в себя две части: кодировщик (encoder) и декодировщик (decoder). Строение одного такого блока представлено на рисунке 2.

Кодировщик преобразует входную последовательность в сжатое представление. Каждый блок кодировщика имеет 3 основных уровня: многоузловой механизм внимания (multi-head attention), слой нормализации и многослойного перцептрона (MLPs). Между подслоями находятся слои нормализации и отсева, а также остаточные соединения между ними.

Целью декодировщика является объедение выходных данных кодировщика с целевой последовательностью и на основе этих данных предсказание следующего токен. Структура декодировщика является аналогичной структуре кодировщика, за исключением наличия третьего компонента, которым является механизм маскированного многоузлового внимания (Masked Multi-Head Attention).

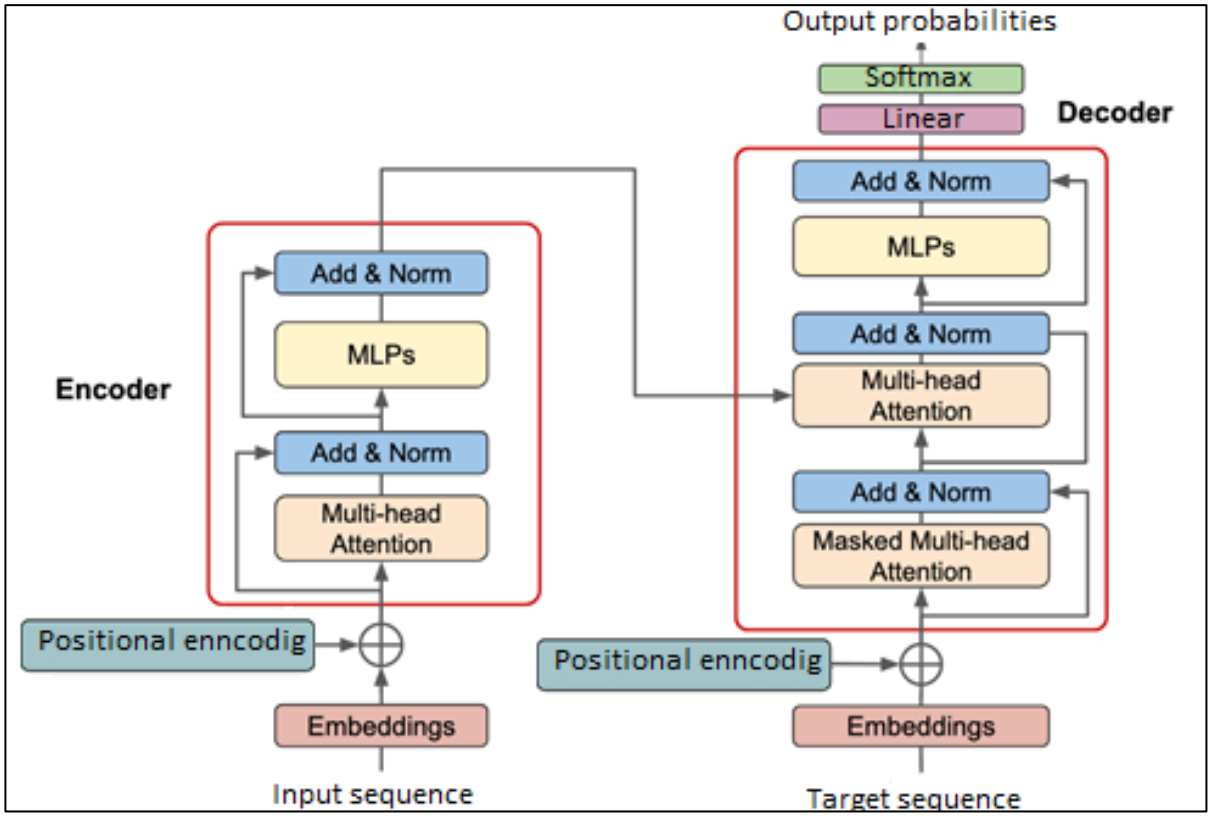


Рисунок 2 – Архитектура блока Transformer

RAG (Retrieval Augmented Generation) или генерация с дополнительным поиском – подход для работы с большими языковыми моделями (LLM), при котором модель строит ответ на вопрос на основе вопроса и найденного дополнительного контекста.

Большие языковые модели (LLM) продемонстрировали замечательные возможности решения задач в широком спектре областей. Важной областью применения являются ответы на вопросы по документам частного предприятия, где основными соображениями являются безопасность данных, что требует приложение, которое может быть развернуто на месте при ограниченных вычислительных ресурсах и корректно отвечает на запросы. Retrieval Augmented Generation (RAG) является наиболее удобной платформой для создания приложений на базе LLM [8].

Когда LLM задают вопросы, относящиеся к конкретной предметной области, за пределами их обучающих данных, они могут генерировать неверную информацию или «галлюцинации» [9]. Использование RAG позволяет обойти данное ограничение посредством извлечения информации из внешнего источника, например, базы данных, и дальнейшей её передачи в качестве контекстной информации в модель LLM для формирования ответов. На рисунке 3 представлена схема RAG архитектуры.

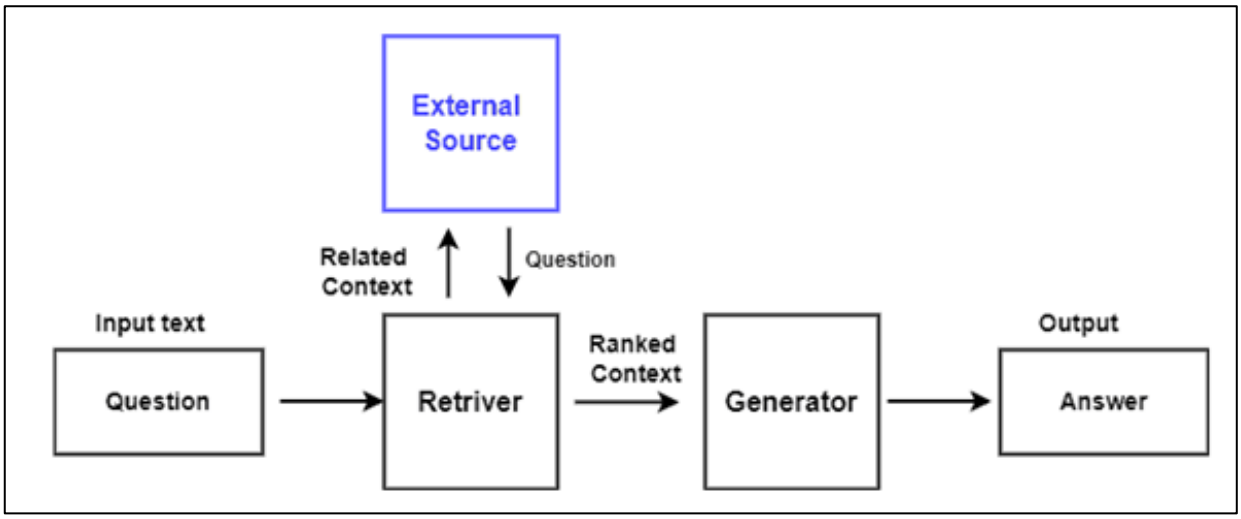


Рисунок 3 – Схема RAG архитектуры

Данную архитектура в своем самом простом варианте состоит из трех компонентов поисковик, внешние данные и генеративная модель. Поисковик (retriever) ищет контекст из внешних данных, которые могут быть представлены в виде векторной базы данных, среди наиболее схожий с текстов с текстом вопроса пользователя. В дальнейшем полученный контекст и текст вопроса передается в генеративную модель, которая строит ответ.

1.2. Выбор модели

Для выбора наиболее подходящей модели для решения поставленной задачи, рассмотрим наиболее популярные модели и сравним их качество при обучении на одинаковом наборе данных.

Для обучения всех моделей был использован датасет SQuAD 2.0 (Stanford Question Answering Dataset 2.0). Данный датасет представляет собой крупномасштабный набор данных для ответов на вопросы, состоящий из вопросов, заданных краудворкерами в наборе статей Википедии, где ответом на каждый вопрос является фрагмент текста, а также вопрос может быть без ответа [10]. А в качестве основных метрик для оценки качества используются F1 и EM scores. Оценка F1 (F-мера) – среднее значение Точности измерений (Accuracy) и Отзыва (Recall) с весами при наличии. Оценка EM (точное совпадение) измеряет процент предсказаний, которые точно соответствуют любому из основных ответов [11].

Модель BERT (Bidirectional Encoder Representations from Transformers) – представляет собой модель машинного обучения (ML) для обработки естественного языка. Он был разработан в 2018 году исследователями Google AI Language и служит в качестве швейцарского армейского ножа для решения более 11 наиболее распространенных языковых задач, таких как анализ настроений и распознавание именованных объектов [12].

Модель BART (Bidirectional and Auto-Regressive Transformer) –Архитектура BART похожа на сеть кодер-декодер, за исключением того, что в ней используется комбинация моделей BERT и GPT, что позволяет сочетать

в себе возможности двунаправленной и авторегрессивной моделей. Авторегрессивная модель предсказывает будущее слово из набора слов с учетом контекста. Модели BART могут быть точно настроены на небольших контролируемых наборах данных для создания задач, специфичных для предметной области [13].

Модель T5 (Text-To-Text Transfer Transformer) – нейросетевая модель для понимания и генерации текста. Отличительная черта данной модели – это использование трансферного обучения, при котором модель сначала предварительно обучается задаче с большим объемом данных, а затем подвергается точной настройке для последующей задачи, что делает ее мощным и эффективным инструментом обработки естественного языка (NLP) [14]. Дообучение позволяет модели решать такие узко направленные задачи, как машинный перевод, классификации текста, суммаризация текстов, генерация текста, составление ответа на вопрос, распознавания именованных сущностей и т.д. Самое главное преимущество данной модели, что Google выпустил две версии T5: первая понимает только английский язык, а вторая является мультиязычной, что позволяет ей понимать 101 язык, в том числе русский язык.

Результаты обучения взяты из следующих исследовательских статей: «Question Answering Models: A Comparison» [11] и «Gestalt: a Stacking Ensemble for SQuAD2.0» [15]. Для удобства сравнения результаты представлены в таблице 1.

Таблица 1 – Сравнение популярных моделей архитектуры Transformer в задаче генерации ответа на вопрос

Модель	**EM scores**	**F1 scores**
BERT	84,3	91,0
BART	88,8	94,6
T5	87,1	90,3

Очевидно, что явным лидером является модель глубокого обучения BART, однако доступ к данной модели осуществляется через сервера Google и имеет прямой доступ к сети интернет, так что дообучение модели

на документации продукта TIP может привести к утечке данных. Поэтому наилучшим вариантом для решения поставленной задачи будет использование модели T5.

1.3. Анализ аналогичных проектов и существующих решений

Так как разрабатываемый чат-бот будет использоваться для ответов на вопросы клиентов по продукту R-Vision TIP, то в качестве аналогичных программных систем рассмотрим несколько узко-направленных чат-ботов использующихся в реальных бизнес решениях.

Так как предметные области данных систем очень сильно отличаются, то для их анализа основными критериями будет служить следующий ряд параметров, таких как функциональность или перечень функций, предоставляемых чат-ботом, удобство и простота использования, качество ответов, насколько точные и информативные ответы на вопросы получает конечный пользователь.

В настоящее время сложно выделить проекты имеющие аналогичную предметную область по теме работы, так как она является узконаправленной и закрытой. Поэтому в рамках работы рассмотрено несколько общедоступных чат-ботов успешно использующихся в бизнес решениях для ответов на вопросы клиентов. К таким относятся следующие системы: чат-бот «НОНА» фабрики «Палитра», робот Макс на государственном портале «Госуслуги», чат-бот от банка «Альфа-банк».

НОНА – Новый Обойный Навигатор [16]

НОНА – это чат-бот обойной фабрики «ПАЛИТРА», появившийся в 2021 году. Данный чат-бот является интерактивным помощником в мире обоев и дизайна.

Функциональные возможности включают в себя следующие пункты:

1) уточнение цены и наличия обоев;
2) помощь в подборе обоев;
3) объяснение условий сотрудничества с фабрикой;

4) поиск обойных магазинов в текущем городе;

5) передача ссылок на рекламные материалы;

6) справка о наличии вакансий.

Что касается пользовательского интерфейса, НОНА является простым и удобным виртуальным помещиком, запросы можно вводить как самостоятельно, так и использовать заготовленные, которые находятся выше поля ввода и представлены в системе в виде кнопок. Пример интерфейса представлен на рисунке 4.

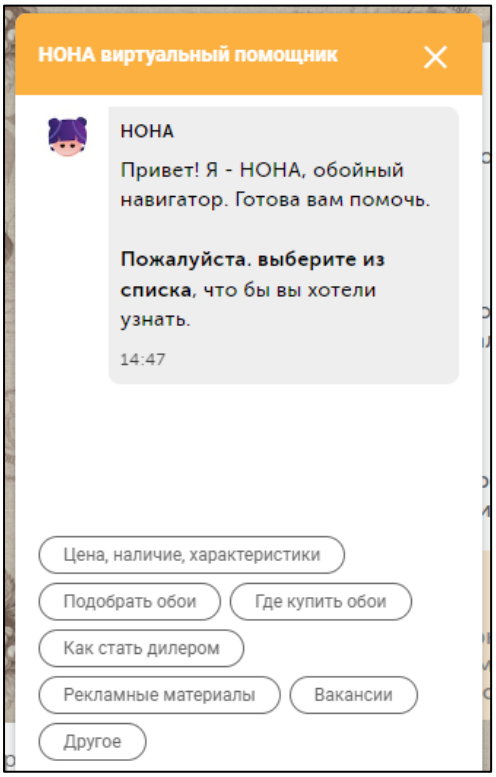


Рисунок 4 – Интерфейс чат-бота «НОНА»

Качество и полнота ответов в целом удовлетворительны, основная проблема возникает при поиске обоев по артикулу: несмотря на то, что в интернет-магазине показывает, что обои имеются в наличии, однако чат-бот не может их найти. Нет проверки вопроса на релевантность.

Данная система относится к чат-ботам, полностью основанным на бизнес-правилах, то есть имеющее дерево диалога, по которому и стоится заранее заданных диалог с пользователем. Отсюда вытекают проблемы с

гибкостью в ответах и сложностью в отслеживании и пополнении информации о наличии товаров.

Робот Макс на государственном портале «Госуслуги» [17]

Робот Макс – цифровой ассистент, который автоматизирует работу поддержки, помогает пользователям найти и подобрать услуги, записаться на прием. Фактически данная система является поисковиком по различным услугам. Для выбора ответа используется информация о длине введенной фразы, способе ввода, намерении пользователя и наличии ответа в сервисе.

Рассматриваемая система позволяет найти услугу, рассказать о ней, объяснить, как действовать, и все это на понятном языке, а также делать запросы в различные государственные ведомства.

Пользовательский интерфейс ассистента является простым и дружественным для новых пользователей. По мере ввода запроса появляются слова подсказки, которые заметно упрощают работу с системой. Как выглядит интерфейс данного чат-бота представлено на рисунке 5.

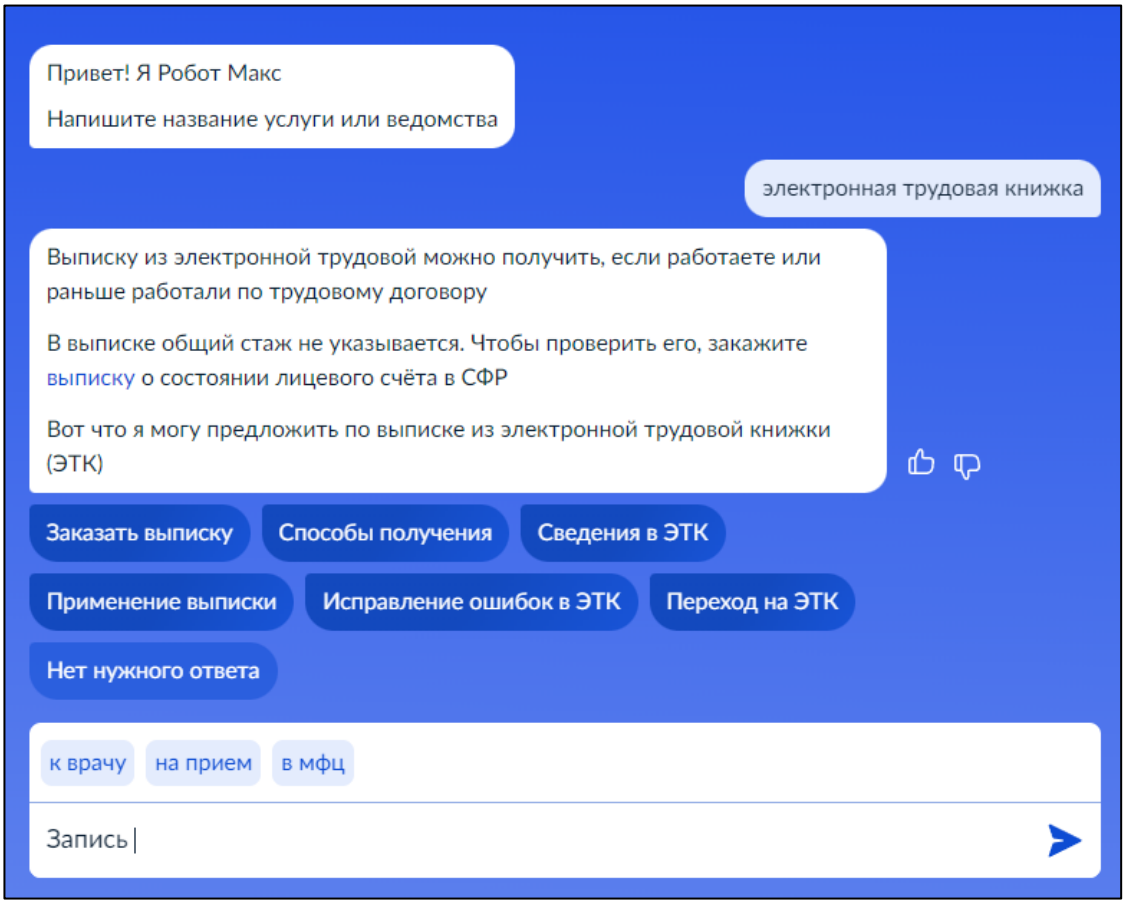


Рисунок 5 – Интерфейс Робота «Макс»

Точность ответов очень высокая, ошибок замечено не было, тем более наличие подсказок сокращает количество ошибок. Также имеете проверка вопросов на релевантность, то есть если ввести нерелевантный вопрос система, то возвращается следующий ответ: «Попробуйте задать вопрос иначе».

Альфа-Помощник [18]

Чат-бот Альфа-Банка умеет оформлять обращение в банк прямо из чата с помощью виджетов. Это удобно для пользователя, не надо переключать экраны или вызывать отдельные функции для обращения. Кроме того, уже сейчас Альфа-Помощник учится распознавать голосовые сообщения и изображения, присланные в чате.

Интерфейс не самый интуитивный, все запросы делаются с помощью голоса или заготовленных шаблонов. Интерфейс голосового помощника представлен на рисунке 6.

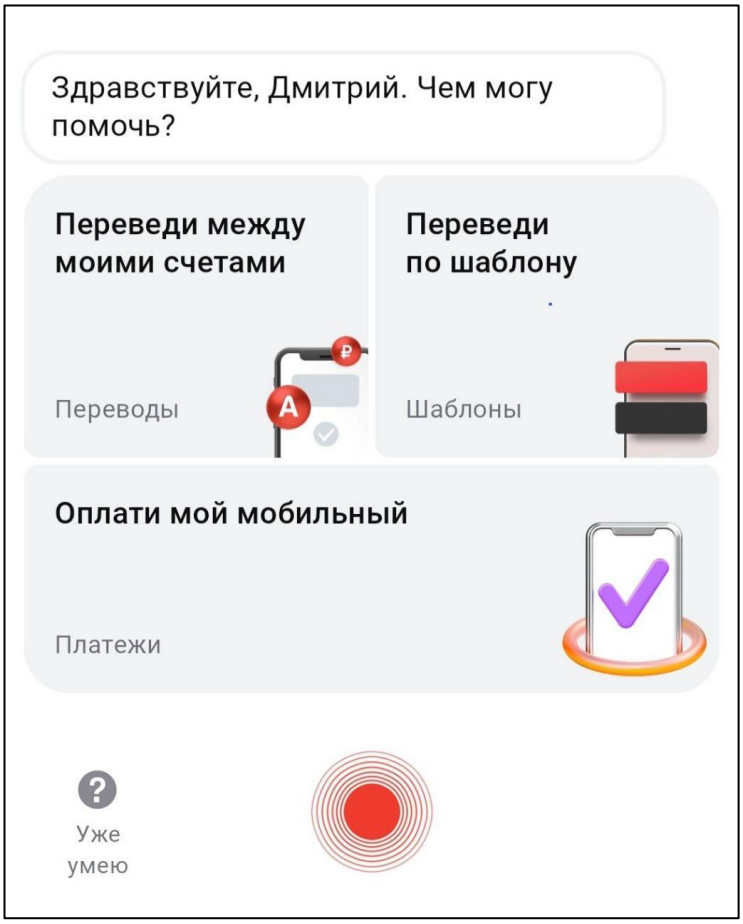


Рисунок 6 – Интерфейс чат-бота от Альфа-банка

Данный чат-бот основан на решении JAICP от компании Just AI из Санкт-Петербурга. Данное решение реализовано на модели обработки естественного языка (NLP), благодаря такому подходу в 2022 году автоматизацию удалось довести до 55%.

Как утверждается в официальной статье от Альфа-банка, точность распознавания запросов превышает 93%, то есть в 93 случаях из 100 алгоритм точно определяет суть запроса без дополнительных уточнений.

На данный чат-бот Альфа-Банка признан лучшим ботом среди банковских помощников страны по итогам рейтинга Chatbot Rank 2023 аналитического агентства Markswebb [19].

Вывод по первой главе

Обзор аналогичных решений показал, что на данный момент существует большое число успешных примеров интеграции чат-ботов в различные продукты. Данная практика позволяет различным компаниям получить заметное конкурентное преимущество, по такому параметру, как создание положительного клиентского опыта.

Также были изучены различные методы реализации системы ответа на вопрос. Архитектура генерации с дополнительным поиском является наиболее современным и гибким подходом в решении задачи построения ответа на вопрос. Данная архитектура позволяет без дополнительного обучения расширить контекст модели и уменьшить число ошибок и «галлюцинаций» LLM с помощью передачи информации, относящейся к теме вопроса.

Среди существующих решений выделяется банковский чат-бот от компании «Альфа-банк». Этот пример показывает, что использование нейронных сетей глубокого обучения действительно является эффективным и современным подходом для реализации чат-бота для решения задачи предоставления ответов на вопросы клиентов.

2. ПРОЕКТИРОВАНИЕ СИСТЕМЫ

2.1. Анализ требований к проектируемой системе

Учитывая анализ аналогичных решений, проведенный в главе 1, можно выделить следующие функциональные требования к системе.

1. Чат-бот должен предоставлять ответ на вопрос, касающийся непосредственно продукта TIP или сферы информационной безопасности.
2. Чат-бот должен иметь механизм обработки нерелевантных вопросов и просит пользователя задать вопрос корректным образом.
3. Общение между чат-ботом и пользователем должно производится посредством обмена тестовыми сообщениями в мессенджере Telegram.

В контексте решаемой задачи, нерелевантным вопросом считается любой текст, не относящийся к продукту TIP или предметной области.

На основе обзора предметной области, проведенного в главе 1, а также учитывая пожелания заказчика – компании R-Vision, были определены следующие нефункциональные требования.

1. Система чат-бота должна быть реализована на языке Python версии 3.10 или выше.
2. Чат-бот должен быть развернут в виде бота в мессенджере Telegram.
3. Чат-бот должен использовать архитектуру глубокой нейронной сети T5.
4. Система чат-бота должна иметь гибкий механизм логирования, чтобы иметь возможность отслеживать возможные возникающие ошибки.
5. Система чат-бота должна быть развернута как Docker-контейнер для удобства сопровождения.
6. Чат-бот должен иметь возможность легкой интеграции в корпоративную инфраструктуру.

2.2. Описание актеров и диаграмма вариантов использования

На основании описанных в пункте 1 функциональных требований разработана диаграмма вариантов использования чат-бота для ответов на вопросы по продукту TIP. Данная диаграмм вариантов использования показана на рисунке 7.

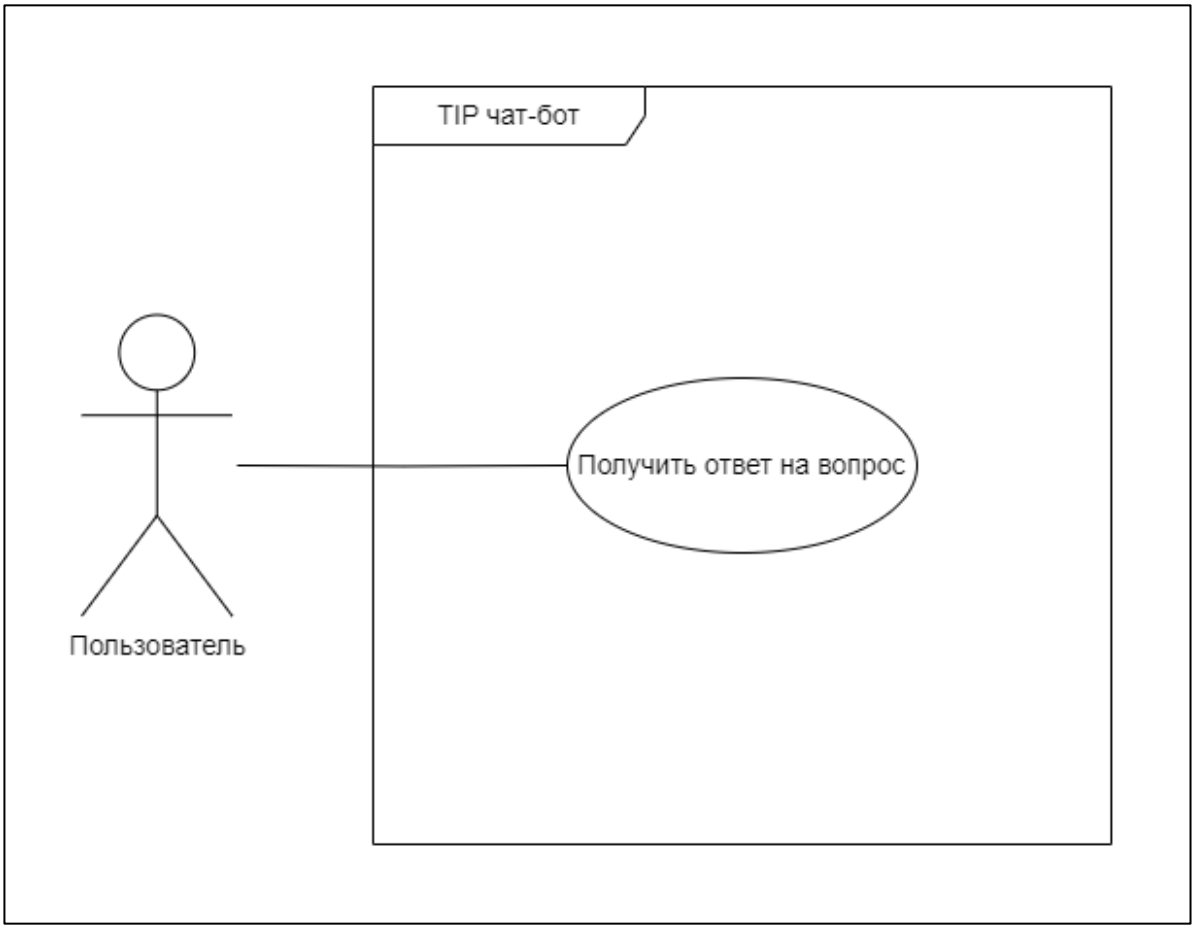


Рисунок 7 – Диаграмма вариантов использования системы

В системе определено только одно действующее лицо (актер) – пользователь. В контексте решаемой задачи пользователь – лицо, которое интересует получение информации о продукте R-Vision TIP, в ближайшем приближении – это клиент пытающийся разобраться в продукте или решить возникшую проблему, а также пользователем может быть новый сотрудник еще неполностью разобравшийся в TIP.

Пользователю доступен один вариант использования системы – «Получить ответ на вопрос». К данному варианту использования была

разработана спецификация, дающая подробное описание прецедента. Данная спецификация представлена в приложении А.

Также спецификация вариантов использования описывает альтернативный поток для варианта использования «Получить ответ на вопрос», который состоит в сообщении пользователю о нерелевантности заданного вопроса.

2.3. Общее описание архитектуры системы

Разрабатываемая система представляет собой чат-бота, решающего задачу ответа на вопросы, посредством отправки пользователем текстового сообщения в мессенджере Telegram и в дальнейшем получения от бота ответа заданный вопрос.

Система построена на архитектуре Retrieval Augmented Generation (RAG) и включает в себя четыре основных компонента вешнее хранилище данных, представленное векторной базой данных, поисковик, генеративная модель и Telegram API для интеграции разработанной RAG в Telegram-бота. Таким образом, система является гибкой, то есть может быть встроена в любую корпоративную среду.

Внешнее хранилище данных представляет собой векторную базу данных, содержащая данные, которые используются в качестве контекста для формирования ответов. В данном решении база данных содержит отдельные тексты документации по продукту R-Vision TIP, а также векторные представления данных текстов, что используются для поиска ближайших соседей с текстом запроса. Использование векторной базы данных, позволяет эффективно хранить данные, а также быстро выполнять операции поиска.

Поисковик или retriever осуществляет процедуры поиска наиболее подходящего контекста во внешнем хранилище данных, а также оценки релевантности запроса пользователя.

Генеративная модель отвечает за создание векторных представлений текста, а также построения ответа на основе вопроса пользователя и найденного контекста. В данном подходе обычно используют большие языковые модели (LLM), которые хорошо себя зарекомендовали в решении подобных задач. Для реализации системы используется модель, основанная на архитектуре Transformer, и заранее пред обучена на текстах русского языка.

Telegram API – элемент обеспечивающий интеграцию разработанной системы в мессенджере Telegram. API позволяет взаимодействовать с пользователем посредствам текстовых сообщений, отправляемых и принимаемых в мессенджере.

На рисунке 2 приложения Д представлена диаграмма деятельности, показывающая происходящие процессы и существующие потоки движения. Рассмотрим данную диаграмму подробнее.

Диалог начинается с отправки пользователем запроса в мессенджере Telegram. Затем в системе последовательно выполняются следующие действия.

1. Посредствам Telegram API система получает сообщение от пользователя.
2. Выполняется предварительная обработка текста запаса.
3. Генеративная модель преобразует текст вопроса в векторное представление.
4. Производится поиск ближайших соседей для текста запроса.
5. Тексты, полеченные из векторной базы данных, проверяются на релевантность посредствам анализа косинусного расстояния между векторами.
6. Если вопрос релевантный, то строится ответ на вопрос с помощью генеративной модели и сформированный текст ответа через Telegram API отправляется пользователю.
7. Если вопрос нерелевантный, то пользователю отправляется сообщение о не корректности заданного вопроса.

2.4. Описание векторной базы данных

Векторные базы данных представляют собой NoSQL-решения, которые хранят данные с помощью векторных вложений. Векторные вложения – это способ представления объектов, таких как предметы, документы или точки данных, в виде векторов. Каждому объекту присваивается вектор, который отражает различные характеристики этого объекта. Эти векторы формируются таким образом, что похожие объекты имеют векторы, расположенные ближе друг к другу в векторном пространстве, в то время как непохожие объекты имеют векторы, расположенные дальше друг от друга [20].

В контексте решаемой задачи объектами, хранящимися в базе данных, являются тексты документации. А в качестве векторной базы данных используется Chroma [21]. Данная база данных состоит из коллекций и хранящихся в них документов. В реализации разрабатываемой системы представлена одна коллекция «vector_db», а каждый документ имеет следующие параметры:

– id – идентификационный номер документа;

– document – текст фрагмента документации;

– embedding – представление текста документации в векторной форме.

Вывод по второй главе

В результате проектирования системы были сформулированы функциональные и нефункциональные требования, предъявляемые к чат-боту. Данные требования послужили основой в выборе архитектуры системы. Также были составлены диаграмма вариантов использования и диаграмма деятельности.

3. РЕАЛИЗАЦИЯ

3.1. Средства разработки

Реализация чат-бота была написана на языке Python 3.10. Для написания кода и его отладки была использована среда разработки PyCharm Professional Edition 2024.1 [22].

Для работы с мессенджером Telegram использовалась библиотека telebot, которая предоставляет интерфейс для взаимодействия с Telegram Bot API [23].

Для работы с большой языковой моделью (LLM) использовалась библиотека transformers. Данная библиотека представляет собой удобный инструмент для использования и обучения моделей в области обработки естественного языка. Разработкой библиотеки занимается компания HuggingFace [23].

Для работы с векторной базой данных была использована библиотека chromadb. Эта библиотека является инструментом для создания векторной базы данных, а также предоставляет инструмента для векторного поиска [21].

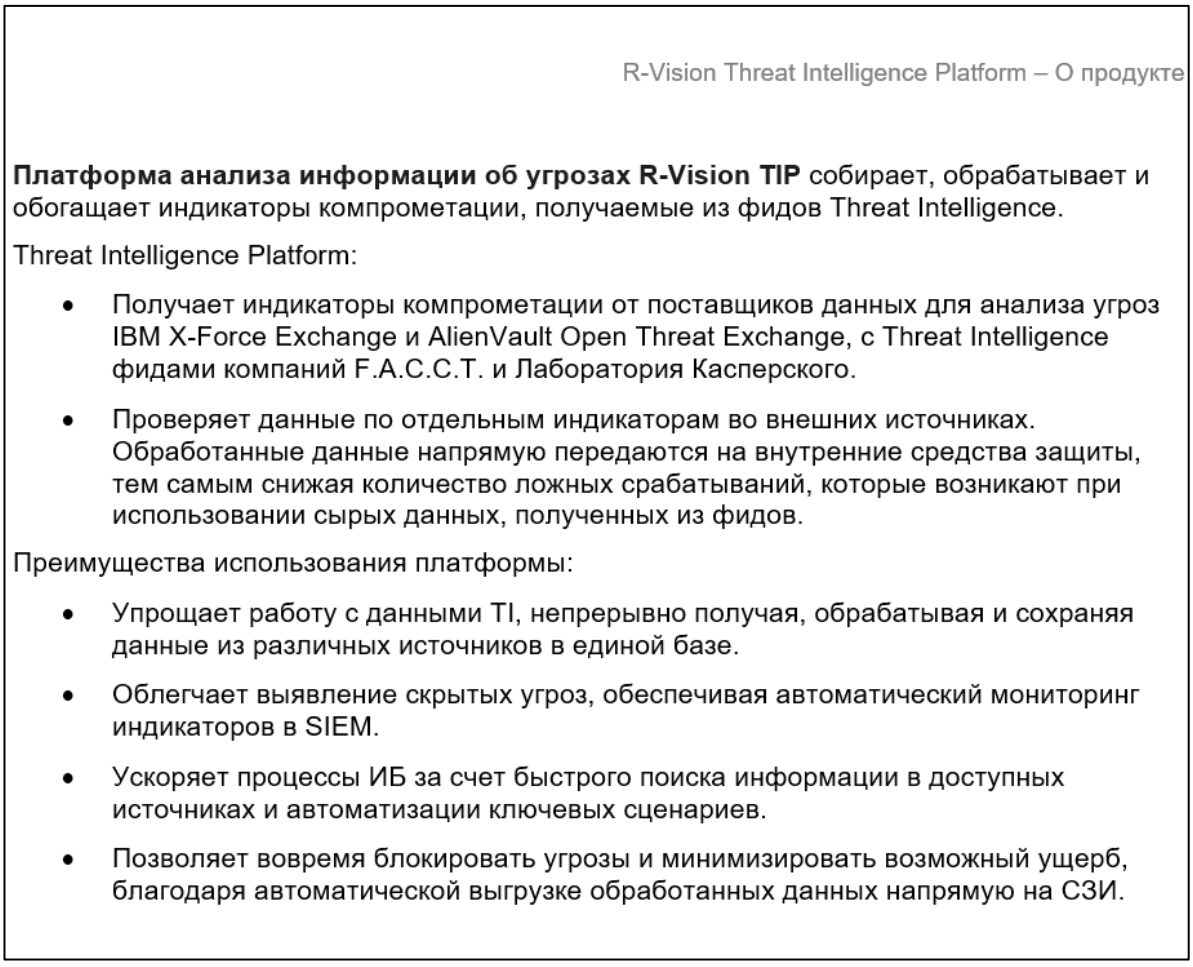
Это лишь основные библиотеки что использовалась для реализации системы, полный список с учетом сторонних зависимостей представлен в файле требований requirements.txt, содержание данного файл продемонстрировано листинге 1 приложения Б. Использование файла требований упрощает процесс развёртывания системы, а также помогает отслеживать уязвимости в сторонних библиотеках.

3.2. Набор данных

Для организации и построения архитектуры генерации с дополнительным поиском (RAG), необходимо собрать набор данных (датасет) для дополнения контекста модели. В контексте решаемой задачи, датасет в подробностях описывает продукт TIP, а также поверхностно охватывает сферу информационной безопасности.

В ходе выполнения работы было составлено три набора данных. Данные датасеты сформированы в ручную на основе документации продукта R-Vision TIP, а также на клиентских вопросах и ответах, взятых из корпоративного чата, однако в целях безопасности, данные из корпоративного чата не используются в демонстрационной версии системы. Для составления датасетов использовался Excel программа из пакета Microsost office [25].

Исходные данные представляют собой документацию продукта TIP, в виде 58 документов в формате docx. На рисунке 8 изображено, как выглядит исходный текст документации. Данные тексты документация разделены тематические главы и разделы, и содержат информацию доступную конечному пользователю.


R-Vision Threat Intelligence Platform – О продукте

Платформа анализа информации об угрозах R-Vision TIP собирает, обрабатывает и обогащает индикаторы компрометации, получаемые из фидов Threat Intelligence.

Threat Intelligence Platform:

- Получает индикаторы компрометации от поставщиков данных для анализа угроз IBM X-Force Exchange и AlienVault Open Threat Exchange, с Threat Intelligence фидами компаний F.A.C.C.T. и Лаборатория Касперского.
- Проверяет данные по отдельным индикаторам во внешних источниках. Обработанные данные напрямую передаются на внутренние средства защиты, тем самым снижая количество ложных срабатываний, которые возникают при использовании сырых данных, полученных из фидов.

Преимущества использования платформы:

- Упрощает работу с данными TI, непрерывно получая, обрабатывая и сохраняя данные из различных источников в единой базе.
- Облегчает выявление скрытых угроз, обеспечивая автоматический мониторинг индикаторов в SIEM.
- Ускоряет процессы ИБ за счет быстрого поиска информации в доступных источниках и автоматизации ключевых сценариев.
- Позволяет вовремя блокировать угрозы и минимизировать возможный ущерб, благодаря автоматической выгрузке обработанных данных напрямую на СЗИ.

Рисунок 8 – Исходный текст документации

Существует несколько подходов составления наборов данных для дополнения контекста, были использованы следующие из них:

– объединение всех текстов в один документ и дальнейшее разбиение на небольшие части (chunk);

– разбиение текстов документации на тематические разделы;

– составление набора вопросов и ответов, охватывающих большой объем документации.

Первый набор данных представляет собой файл в формате txt, содержащий полных объем всех текстов документации. На рисунке 9 представлен фрагмент этого набора данных.

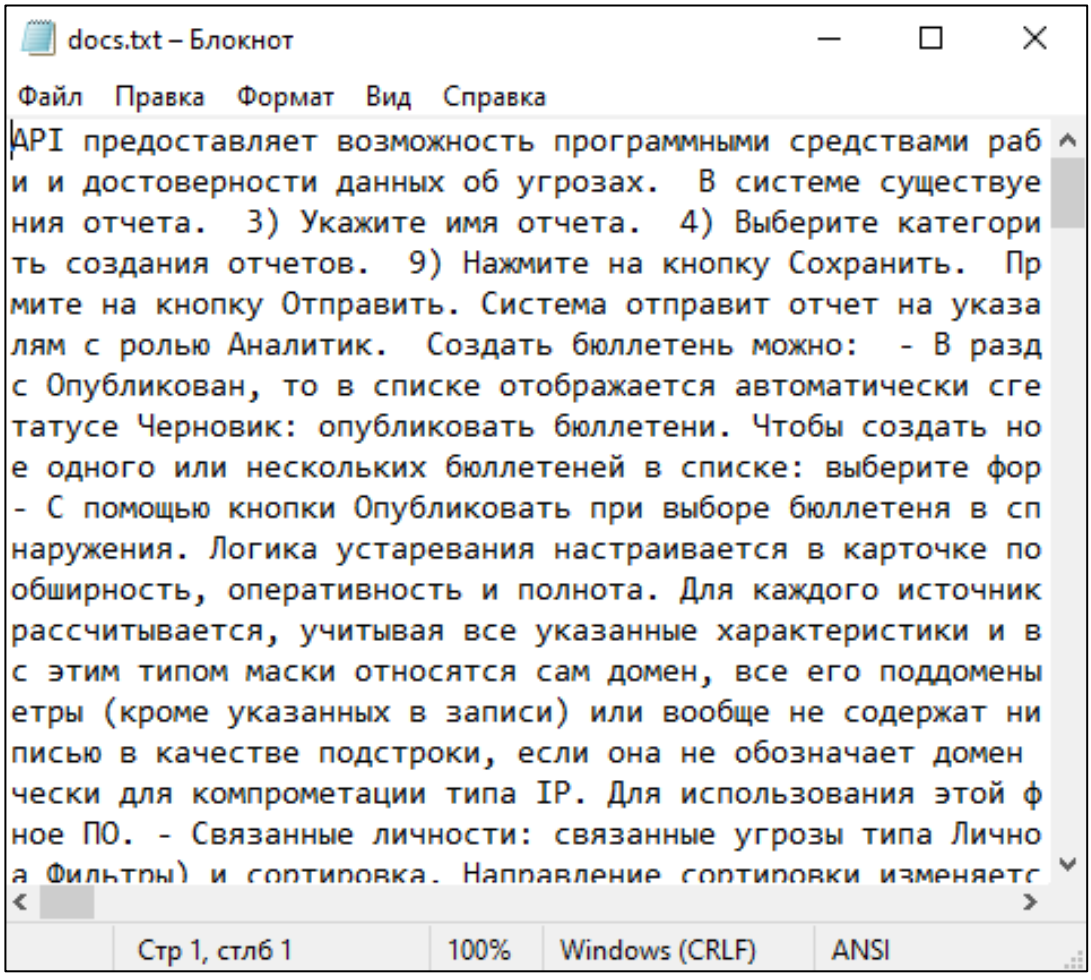


Рисунок 9 – Объединение всех текстов документации

Следующий датасет упорядочивает документацию продукта TIP, разбивая длинные главы на более понятные части. Датасет представлен в виде

csv-файла, который состоит из 5 столбцов и 289 строк. Каждая запись содержит следующую информацию:

– id – идентификатор строки;

– doc_id – идентификатор текста документации;

– title – название главы документации;

– head – подзаголовок небольшой объединенной по смыслу части документации;

– doc_text – текст объединенной по смыслу части документации.

На рисунке 10 представлен вид данного набора данных.

id	doc_id	title	head	doc_text
0	0	Пользова	Использо	API предоставляет воз
1	0	Пользова	Авторизаі	Для авторизации в API
2	1	Пользова	Об анали	Аналитические отчеты
3	1	Пользова	Просмотр	Список аналитических
4	1	Пользова	Добавлен	Добавлять и редактир
5	1	Пользова	Автомати	Добавлять и редактир
6	1	Пользова	Экспорт а	Отчет можно экспорти
7	1	Пользова	Отправка	Отчет можно отправит
8	2	Бюллетен	Что такое	В системе можно созда
9	2	Бюллетен	О создани	Создание и редактиро
10	2	Бюллетен	Как созда	Чтобы создать бюллет
11	2	Бюллетен	Просмотр	Список бюллетеней от

Рисунок 10 – Разбиение документации на разделы и главы

Третий набор данных – это перечень вопросов и ответов, охватывающих каждый раздел документации. Набор данных также имеет csv формат, состоящий из 3 столбцов и 1180 строк. Каждая строка содержит следующую информацию:

– question – вопрос;

– answer – ответ;

– doc_id – идентификатор, соответствующий строке документации из предыдущего набора данных.

На рисунке 11 представлен фрагмент датасета, состоящего из вопросов и ответов.

question	answer	doc_id
Какие мет	Для взаим	0
Где можн	Актуальна	0
Как осуще	Взаимоде	0
Какие про	Для взаим	0
Где наход	Информаі	0
В каком ф	Токен для	1
Как перед	Для перед	1
Где можн	Токен для	1
Какие пра	При выпо	1
Как опред	Определе	1
Какую инс	Аналитиче	2
Какие фор	Аналитиче	2

Рисунок 11 – Вопросы и ответы по документации R-Vision TIP

3.3. Регистрация чат-бота

Перед разработкой, необходимо зарегистрировать чат-бота и получить уникальный id, являющийся одновременно и токеном для подключения. Для этого в Telegram существует специальный бот BotFather.

Для регистрации Telegram чат-бота достаточно зайти в бот BotFather, прописать команду /newbot и следующим сообщением задать имя нового бота, единственное ограничение, имя обязательно должно оканчиваться на «bot». В случае успеха BotFather возвращает токен бота, иначе нужно попробовать задать другое имя для будущего чат-бота. После регистрации бот создан, однако в нем не реализован никакой функционал, помимо базового, который предоставляет мессенджер Telegram. На рисунке 1 приложения Г представлен процесс регистрации чат-бота.

3.4. Реализация компонентов системы

Реализация чат-бота выполнена для мессенджера Telegram, однако основная часть системы не зависит от части подключения к мессенджеру, это значит, что данное решение без сложностей может быть интегрировано в любой мессенджер, в том числе и в корпоративную систему. Реализованная система построена на архитектуре RAG (Retrieval Augmented Generation) и состоит из трех основных компонентов: генеративной модели, векторной базы данных и векторного поисковика.

Генеративная модель

В качестве генеративной модели используется готовая большая языковая модель, построенная на архитектуре Transformers. Данная LLM является дообученной русскоязычной моделью T5 от Сбербанка: FRED-T5-LARGE_text_qa [26].

Весь необходимых функционал для работы с моделью был реализован в обработчике модели – `ModelHandler`. Данный обработчик представляет собой класс, имеющий следующие поля:

– `use_cuda` – булевое значение, определяющее возможность использования графического процессора для работы модели;

– `device` – определяет какое устройство будет использоваться для вычислений модели;

– `generation_config` – представляет собой набор параметров генерации;

– `tokenizer` – токенизатор, который отвечает за подготовку входных данных для модели;

– `model` – модель, заранее предобученная на русскоязычых текстах.

Поля `generation_config`, `tokenizer`, `model` загружаются из библиотеки transformers при инициализации `ModelHandler`.

Также для работы с моделью обработчик имеет методы:

– `get_embeddings` – преобразует текст в векторное преставление;

– `generate` – генерирует ответ по составленному промпту.

Промпт представляет собой подсказку для модели, помогающую составить ответ. Для данной модели промпт имеет следующую структуру, открывающий и закрывающий теги модели: `<SC6>` и `<extra_id_0>` соответственно. После открывающего тега идет слово «Текст:» и за ним подставляется контекст, полученный от поисковика по векторной базе данных, со следующей строки пишется вопрос от пользователя. В конце со следующей строки прописывается «Ответ:». Данная структура составления промпта продемонстрирована в листинге 1. А листинг 2 содержит код обработчика модели.

Листинг 1 – Структура промпта генеративной модели

```
prompt = f'''<SC6>Текст: {context}\nВопрос: {question}\nОтвет:
            <extra_id_0>'''
```

Листинг 2 – Обработчик модели `ModelHandler`

```
class ModelHandler:
    def __init__(self):
        self.use_cuda = torch.cuda.is_available()
        self.device = torch.device("cuda" if self.use_cuda else "cpu")
        self.generation_config =
GenerationConfig.from_pretrained("Den4ikAI/FRED-T5-LARGE_text_qa")
        self.tokenizer = AutoTokenizer.from_pretrained("Den4ikAI/FRED-T5-
LARGE_text_qa")
        self.model = AutoModelForSeq2SeqLM.from_pretrained("Den4ikAI/FRED-
T5-LARGE_text_qa").to(self.device)
        self.model.eval()

    def get_embeddings(self, text: str) -> list[float]:
        """
        Convert text to embedding

        :param text: input text
        :return: embeddings
        """
        enc = self.tokenizer(text, return_tensors="pt")
        output = self.model.encoder(
            input_ids=enc["input_ids"],
            attention_mask=enc["attention_mask"],
            return_dict=True
        )

        emb = output.last_hidden_state[:, 0]
        return emb.detach().numpy().tolist()[0]

    def generate(self, prompt: str):
        """
        Generates text using a prompt

        :param prompt: Compiled prompt
```

```
        :return: generated text
        """
        data = self.tokenizer(f"{prompt}", return_tensors="pt")
        output_ids = self.model.generate(
            **data,
            generation_config=self.generation_config)[0]
        out = self.tokenizer.decode(output_ids.tolist())
        return out
```

Поисковик и векторная база данных

Два следующих компонента системы тесно связаны, поэтому функционал работы с векторной базой данных и поисковик были объединены в обработчике базы данных – `ChromaHandler`, данный класс приведен в листинге 2 приложения Б. Данный обработчик при инициализации в качестве аргумента получает экземпляр класса `ModelHandler`, а также содержит следующие поля:

– `model` – экземпляр класса `ModelHandler`;

– `chroma_client` – экземпляр класса `ChromaClient`, позволяющий работать с векторной базой данных.

Также `ChromaHandler` имеет методы:

– `_load_documents` – загружает тексты документации из датасета, предобрабатывает их и приводит их к необходимому виду;

– `save_to_chroma_db` – создает коллекцию и сохраняет в нее все документы полученные из датасета;

– `get_close_texts` – производит по коллекции поиск текстов наиболее близких по косинусному расстоянию с текстом вопроса.

3.5. Предобработка текстов

Для предобработки текстов был разработан отдельный файл, содержащий функции для удаления и замены некорректных символов. Для этого используются регулярные выражения и фильтрация. В листинге 3 представлены данные функции.

Листинг 3 – Предобработка текстов

```
def remove_not_ascii(text: str) -> str:
    """
```

```
    Remove non-ASCII characters from a string

    :param text: text
    :return: processed text
    """
    cyrillic_str_lower = 'абвгдеёжзийклмнопрстуфхцчшщъыьэюя'
    cyrillic_str_upper = cyrillic_str_lower.upper()
    ascii_chars = string.printable
    chars = set(cyrillic_str_lower + cyrillic_str_upper + ascii_chars)

    return ''.join(filter(lambda x: x in chars, text))

def remove_invalid_chars(text: str) -> str:
    """
    Remove invalid symbols from text
    :param text: text
    :return: processed text
    """
    text = re.sub(r'’', "\'", text)
    text = re.sub(r'`', "\'", text)
    text = re.sub(r'‟', '\"', text)
    text = re.sub(r'″', '\"', text)
    text = re.sub(r' {2}', ' ', text)

    return text


def clean_text(text: str) -> str:
    """
    Remove non-ASCII and invalid characters from string

    :param text: text
    :return: processed text
    """
    text = remove_invalid_chars(text)
    text = remove_not_ascii(text)
    return text
```

Также здесь содержатся функции для удаления тегов модели, использующая регулярные выражения, программная реализация представлена в листинге 4.

Листинг 4 – Удаление тегов модели

```
def remove_model_tags(text) -> str:
    """
    Remove model tags from text

    :param text: generated text
    :return: cleaned text
    """
    return re.sub(r'\<.*?\>', '', text)
```

Реализация отсева нерелевантных текстов документации также расположена в данном файле. Для отсева используется функция `text_similarity`, которая вычисляет косинусное расстояние между

векторными представлениями текстов вопроса и документации, возвращает данное значение. Булевая функция `is_similar_texts` возвращает True если векторное расстояние больше или равно пороговому значению, в противном случае False. Экспериментально было выяснено что наилучшее пороговое значение расположено на уровне 0,5. В листинге 5 содержатся данные функции.

Листинг 5 – Отсев нерелевантных текстов

```
def text_similarity(question: str, text: str, model) -> float:
    """
    Calculate similarity between question and text

    :param question: question text
    :param text: context text
    :param model: model to calculate similarity
    :return: calculated similarity
    """
    encoding1 = model.get_embeddings(question.lower())
    encoding2 = model.get_embeddings(text.lower())

    similarity = np.dot(encoding1, encoding2) / (np.linalg.norm(encoding1)
* np.linalg.norm(encoding2))

    return similarity
def is_similar_texts(question: str, text: str, model) -> bool:
    """
    Check if text is similar to question

    :param question: question text
    :param text: context text
    :param model: model object
    :return: True if text is similar to question, else False
    """
    return text_similarity(question, text, model) >= 0.5
```

Также была реализована функция для обобщения текста, что использовалась только в экспериментах, так как эксперименты показали, что использование обобщения при сохранении текстов документации снижает качество итоговых ответов. Для обобщения использовалась библиотека `TextRank`. Функция обобщения текста представлена в листинге 6.

Листинг 6 – Обобщение текста

```
def get_summary(text: str) -> str:
    """
    Get summary from text
    :param text: text to be summarized
    :return: string summary of text
    """
    parser = PlaintextParser.from_string(text, Tokenizer("russian"))
    summarizer = TextRankSummarizer()
```

```
    summary = summarizer(parser.document, 2)
    text_summary = ""
    for sentence in summary:
        text_summary += str(sentence)
    return text_summary
```

3.6. Реализация API

Для связи основной части системы с мессенджером Telegram используется библиотека telebot, предоставляющая интерфейс для взаимодействия с Telegram Bot API.

Подключение к определенному чат боту производится с посредством уникального токена, который выдается при создании бота. Этот токен записан в файле `settings.py`, как `BOT_ID`. Полный код файла настроек, содержащего основные параметры системы, приведен в листинге 7.

Листинг 7 – Файл настроек

```
import os

CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
SOURCE_DATA_DIR = os.path.join(CURRENT_DIR, "source_data")

VEC_MODEL_PATH = os.path.join(SOURCE_DATA_DIR, "vec_model",
"word_vectors.w2v")


CHROMA_NAME = "vector_db"
CHROMA_PATH = os.path.join(CURRENT_DIR, "db")
DATA_PATH = os.path.join(SOURCE_DATA_DIR, "qa_dataset.csv")

CHUNK_SIZE = 750
CHUNK_OVERLAP = 300

BOT_ID = '******************************************'
```

Инициализируется объект класса `TeleBot`. А также для работы с системой инициализируются обработчики `ModelHandler` и `ChromaHandler`.

Дальше реализуются обработчики сообщений, они представляют собой функцию обернутую в декоратор (функция, которая принимает другую функцию в качестве аргумента, добавляют к ней некоторую дополнительную функциональность и возвращают функцию с измененным поведением). Данная программная реализация представлена файле `app.py`, а также код показан в листинге 8.

Листинг 8 – Реализация Telegram API

```
MODEL = ModelHandler()
DB = ChromaHandler(MODEL)
BOT = telebot.TeleBot(BOT_ID)

@BOT.message_handler(commands=['start'])
def start_bot(message):
    username = message.from_user.first_name
    if message.from_user.last_name:
        username += f" {message.from_user.last_name}"
    first_message = f"<b> {username}" \
                    f"</b>, привет!\nДанный бот был создан для ответа на
вопросы по продукту R-Vision TIP"
    markup = types.InlineKeyboardMarkup()
    button_yes = types.InlineKeyboardButton(text='Продолжить',
callback_data='start')
    markup.add(button_yes)
    BOT.send_message(message.chat.id, first_message, parse_mode='html',
reply_markup=markup)

@BOT.message_handler(func=lambda message: True)
def get_text_messages(message):
    question = message.text
    similar_texts = DB.get_close_texts(question)
    if similar_texts:
        context = ''.join(similar_texts)

        prompt = f'''<SC6>Текст: {context}\nВопрос: {question}\nОтвет:
<extra_id_0>'''

        answer = MODEL.generate(prompt)

        BOT.send_message(message.from_user.id, remove_model_tags(answer))
    else:
        BOT.send_message(message.from_user.id, "Некорректный вопрос")

@BOT.callback_query_handler(func=lambda call: True)
def response(function_call):
    if function_call.message:
        if function_call.data == "start":
            second_mess = "Задайте интересующий вас вопрос:"
            BOT.send_message(function_call.message.chat.id, second_mess)
            BOT.answer_callback_query(function_call.id)

BOT.infinity_polling(none_stop=True, interval=0)
```

Функция `start_bot` отвечает за начало общения с ботом. Выводится приветствие, а также пояснение для чего служит данный бот.

Функция `response` сообщения отвечает за отправку сообщения с предложением задать вопрос.

В функции `get_text_messages` происходит предобработка текста вопроса, сформирование и отправка ответа.

Метод `infinity_polling(none_stop=True, interval=0)` запускает бота в режиме бесконечного ожидания новых сообщений, то есть от будет постоянно проверять наличие новых сообщений и обрабатывать их. Параметр `none_stop=True` указывает, что бот будет работать даже в случае ошибок или проблем с подключением.

Для развертывания чат-бота, как изолированного приложения в docker-контейнере разработан `Dockerfile`, который описывает образ контейнера. Базовым образом является `python:3.10.12-slim-buster` – это легковесный образ на основе Debian ОС с интерпретатором Python версии 3.10.12. Далее описываются команды для обновления пакетного менеджера Debian, установка необходимых библиотек Python, а также копирование файлов проекта. Далее следует запуск исполняемого файла с реализованным API – app.py. Написанный Dockerfile системы приведен в листинге 9.

Листинг 9 – Dockerfile сисемы

```
FROM python:3.10.12-slim-buster

RUN apt-get update
RUN pip3 install --upgrade pip

COPY requirements.txt .
RUN pip3 install -r requirements.txt
RUN pip3 install pysqlite3-binary
COPY . .

CMD python app.py
```

Также для удобства развертывания контейнера, был создан файл `docker-compose.yml`. Данный файл представляет собой конфигурационный файл Docker Compose, описывающий параметры запуска docker-контейнера. Содержимое файла `docker-compose.yml` отражено в листинге 10.

Листинг 10 – Файл docker-compose.yml

```
version: '3'
services:
  tip-bot:
    image: tip-bot:latest
```

3.7. Эксперименты для нахождения оптимальных параметров системы

Входе работы были проведены различные эксперименты для определения наилучшей конфигурации системы. Для оценки качества ответов на вопросы были использованы такие метрики, как BLEU и ROUGE. Данные метрики широко применяются для оценки качества моделей обработки естественного языка [27].

BLEU (Bilingual Evaluation Understudy) – данный алгоритм изначально разрабатывалась для оценки качества в такой классической задачи NLP, как машинный перевод текса. Данная оценка рассчитывается для отдельных переведенных фрагментов – как правило, предложений – путем сравнения n-граммов предложений (последовательности из n элементов), переведенных машинным способом, с n-граммами предложений, переведенных человеком. Затем эти оценки усредняются по всему тексту, чтобы оценить общее качество перевода. Ни разборчивость, ни грамматическая правильность не принимаются во внимание. В результате BLEU всегда выдает число от 0 до 1. Это значение указывает, насколько текст-кандидат похож на тексты-ссылки, при этом значения, близкие к 1, представляют большее количество похожих текстов [28].

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) – Этот алгоритм также был создан для оценки машинного перевода. Этот показатель основан на вычислении синтаксического совпадения между полученным и ожидаемым текстом [29]. Для полноты оценки обычно вычисляют следующие значения:

– Rouge-1: показатель, отражающий совпадение униграмм (отдельных слов) между фрагментами текстов;

– Rouge-2: показывает совпадение биграмм (пар слов) между фрагментами текст;

– Rouge-L: вычисляет перекрытие наиболее длинных совпадающих в последовательности n-грамм между фрагментами тексов;

– ROUGE-Lsum метод расчета ROUGE-L применяется на уровне предложения, а затем все результаты суммируются для получения итоговой оценки.

Для проведения экспериментов был составлен датасет содержащий 200 вопросов и ответов. Также разработан скрипт для проверки точности ответов данный скрипт приведен в листинге 3 приложения Б. Данный скрипт полностью отображает работу системы, за исключением взаимодействия с API Telegram, так как вопросы подаются из заготовленного датасета, а ответы сравниваются с готовыми ответами, которые составил человек.

Входе различных экспериментов с предобработкой текста и способами представления данных в векторной базе данных была составлена таблица 2.

Таблица 2 – Результаты экспериментов

№	Rouge1	Rouge2	RougeL	RougeLsum	Blue
1	0,1593	0,0806	0,1568	0,1569	0,0988
2	0,0827	0,0371	0,0791	0,0790	0,0786
3	0,1513	0,0715	0,1490	0,1504	0,0779
4	0,1773	0,0754	0,1748	0,1761	0,0984
5	0,1535	0,0684	0,1512	0,1508	0,0701
6	0,2428	0,1330	0,2420	0,2401	0,2610
7	0,2574	0,1478	0,2528	**0,2554**	**0,3126**
8	**0,2594**	**0,1520**	**0,2574**	0,2569	0,2541

В данной таблице приведены результаты следующих экспериментов.

1. Вся документация разделена на чанки размера 500 и перекрытием в 150 слов, применяется предобработка, как текстов документации, так и текста вопроса.

2. Документация разделена на чанки 300 и перекрытием в 100 слов, применяется предобработка, как текстов документации, так и текста вопроса.

3. Документация разбита на тематические разделы, без применения предобработки.

4. Документация разбита на тематические разделы, применяется предобработка, как текстов документации, так и текста вопроса.

5. Документация разбита на тематические разделы, применяется предобработка, как текстов документации, так и текста вопроса, а также обобщение для текстов документации.

6. Для составления промпта используются готовые пары ответов и вопросов, без пересечения с тестовой выборкой, предобработка не применяется.

7. Для составления промпта используются готовые пары ответов и вопросов, без пересечения с тестовой выборкой, применяется предобработка для текста ответа.

8. Для составления промпта используются готовые пары ответов и вопросов, без пересечения с тестовой выборкой применяется предобработка, как для текста вопроса, так и для текста ответа.

По итогам экспериментов было выявлено, что наилучшим подходом является использование готовых пар вопрос-ответ для сформирования контекста, с предобработкой как текста промпта, так и текста вопроса для модели. А наименее хороший результат показывает разбиение целого текста на чанки, также выяснялось данный подход очень чувствителен к изменению длины чанка и допустимого пересечения. Разбиение текста на тематические главы и разделы показывает схожие результаты с предыдущим подходом, однако данный метод имеет следующее преимущество – простота дополнения и обновления датасета актуальной информацией

Вывод по третьей главе

В рамках данной главы были описаны средства разработки, процесс подготовки наборов данных, разработки основных компонентов системы и API для связи бота мессенджера Telegram. А также был подобран оптимальный подход составления промпта.

4. ТЕСТИРОВАНИЕ

4.1. Модульное тестирование

Модульное или Unit-тестирование представляет собой тесты, которые направлены на проверку функциональности и работоспособности отдельных модулей исходного приложения [30]. Для разрабатываемой системы были реализованы unit-тесты для каждой функции предобработки текста, а именно `remove_not_ascii`, `remove_invalid_chars`, `clean_text`, `text_similarity`, `is_similar_texts`, `get_summary`, `remove_model_tags`. Данная программная реализация приведена в листинге 4 приложения В, а на рисунке 12 представлен результат работы тестирования, все тесты успешно пройдены.

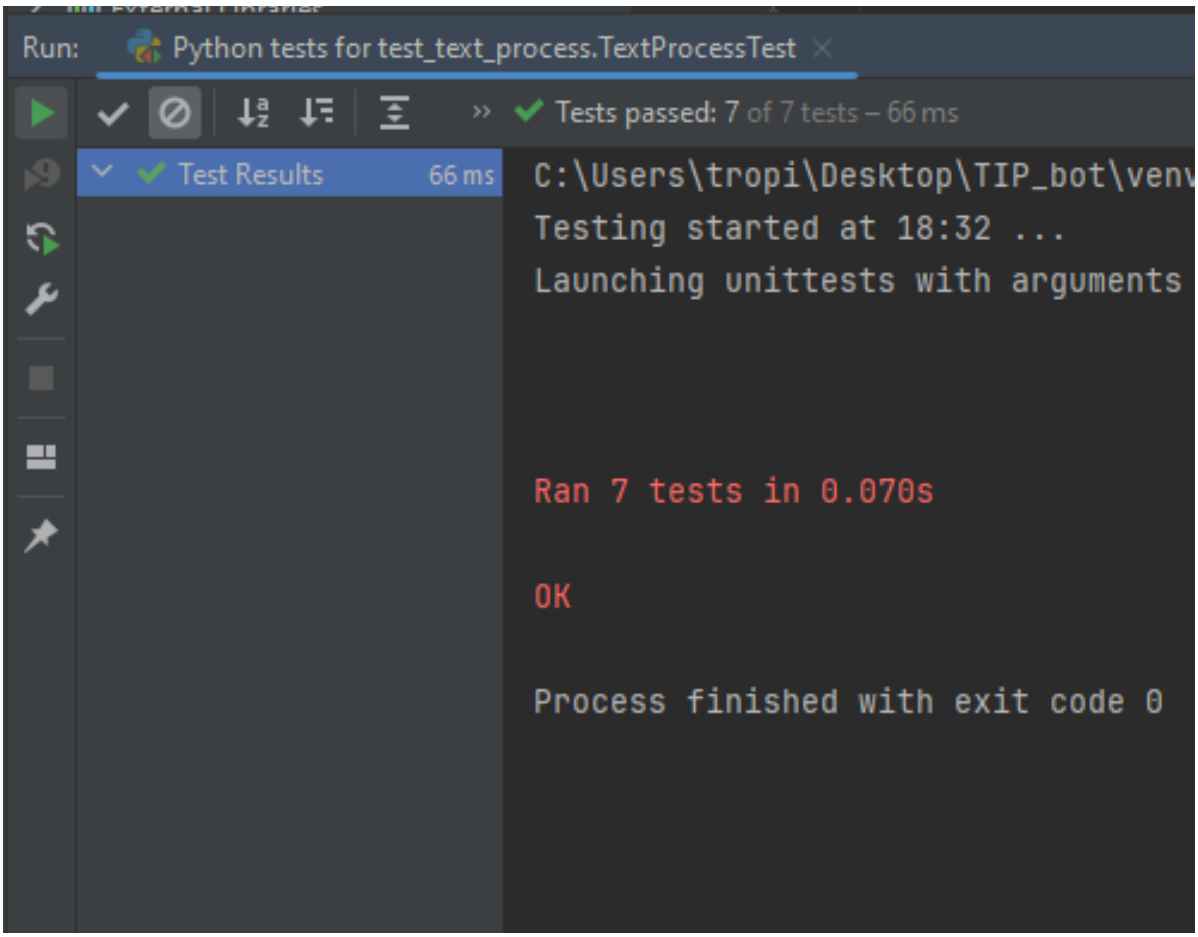


Рисунок 12 – Результаты unit-тестирования

4.2. Функциональное тестирование

После реализации Telegram API было проведено функциональное тестирование, согласно требованиям разработанным на этапе проектирования. Для каждого теста есть только одно предусловие чат-бот запущен. Результаты функционального тестирования отражены в таблице 3.

Таблица 3 – Результаты функционального тестирования

№	Действие	Ожидаемый результат	Тест пройден?
1	Пользователь через мессенджер Telegram отправляет релевантный вопрос.	Чат-бот отвечает пользователю, отправляя ответ на заданный вопрос.	Да
2	Пользователь через мессенджер Telegram отправляет нерелевантный вопрос.	Чат-бот отвечает пользователю, сообщением о не корректности вопроса.	Да
3	Пользователь через мессенджер Telegram, отправляет картинку или стикер.	Чат-бот не отвечает пользователю.	Да
4	Пользователь через мессенджер Telegram, отправляет видео или видеосообщение.	Чат-бот не отвечает пользователю.	Да
5	Пользователь через мессенджер Telegram, отправляет аудио или аудиосообщение.	Чат-бот не отвечает пользователю.	Да

Вывод по четвертой главе

В ходе unit-тестирования было выявлено, что предобработка текста работает корректно. Результаты функционального тестирования показывает, что реализованный чат-бот соответствует поставленным при проектировании требованиям.

ЗАКЛЮЧЕНИЕ

В рамках квалификационной работы был разработан чат-бот для ответов на вопросы клиентов R-Vision TIP.

В ходе работы были решены все поставленные задачи:

1) проведен анализ предметной области и обзор существующих решений;

2) спроектирован и реализован чат-бот;

3) проведен ряд экспериментов для подбора оптимальных пара-метров системы.

4) проведено тестирование разработанного чат-бота.

Данный чат-бот был разработан по инициативе заказчика – компании R-Vision. В дальнейшем планируется внедрение данного решения в инфраструктуру компании R-Vision.

Что касается развития данного проекта, планируются следующие изменения:

1) заменить генеративную модель на более тяжелую, дающую более точные ответы;

2) переписать API для внедрения системы в корпоративную среду;

3) добавить асинхронность обработки запросов;

4) дополнить выборку ответами на вопросы клиентов из корпоративного чата;

5) сохранять вопросы пользователей и ответы для оценки работы системы, а также возможности составить набор данных для дообучения модели.

ЛИТЕРАТУРА

1. Шумилина М.А., Коробко А.В. Разработка чат-бота на языке программирования Python в мессенджере «Telegram». // Научные известия, 2022. – №28. – С. 47–54.

2. Жеребцова Ю.А., Чижик А.В. Сравнение моделей векторного представления текстов в задаче создания чат-бота. // Вестник НГУ, Серия: Лингвистика и межкультурная коммуникация, 2020. – №3. – С. 16–34.

3. Ураев Д.А. Классификация и методы создания чат-бот приложений. // International scientific review, 2019. – №LXIV. – С. 30–33.

4. Yinghui Li, Yong Jiang, Yangning Li, Xingyu Lu, Pengjun Xie, Ying Shen, Hai-Tao Zheng. Bidirectional End-to-End Learning of Retriever-Reader Paradigm for Entity Linking. // Knowledge-Based Systems. 2024. – 27 с.

5. Puri R., Spring R., Patwary M., Shoeybi M., Catanzaro B. Training Question Answering Models From Synthetic Data. // EMNLP, 2020. – С. 5811–5826.

6. How do Transformers work? [Электронный ресурс] URL: https://huggingface.co/learn/nlp-course/chapter1/4 (дата обращения 29.01.2024 г.).

7. Vaswani A., Shazeer N., Parmar N., Uszkoreit J., Jones L., Gomez A.N., Kaiser L., Polosukhin I. Attention is all you need. // Advances in neural information processing systems, 2017. – 30 с.

8. Masoomali Fatehkia, Ji Kim Lucas, Sanjay Chawla. T-RAG: Lessons from the LLM Trenches. // Qatar Computing Research Institute. 2024. – 21 с.

9. Yue Zhang, Yafu Li, Leyang Cui, Deng Cai, Lemao Liu, Tingchen Fu, Xinting Huang, Enbo Zhao, Yu Zhang, Yulong Chen, Longyue Wang, Anh Tuan Luu, Wei Bi, Freda Shi, Shuming Shi. Siren's Song in the AI Ocean: A Survey on Hallucination in Large Language Models. // Tencent AI lab. 2023. – 33 с.

10. Pranav Rajpurkar, Robin Jia, Percy Liang. Know What You Don't Know: Unanswerable Questions for SQuAD. // Annual Meeting of the Association for Computational Linguistics. 2018. – 9 с.

11. Abhijith Neil Abraham. Question Answering Models: A Comparison. [Электронный ресурс] URL: https://blog.paperspace.com/question-answering-models-a-comparison/ (дата обращения: 10.02.2024 г.).

12. BERT 101 State Of The Art NLP Model Explained. [Электронный ресурс] URL: https://huggingface.co/blog/bert-101 (дата обращения: 10.02.2024 г.).

13. BART Model for Text Auto Completion in NLP. [Электронный ресурс] URL: https://www.geeksforgeeks.org/bart-model-for-text-auto-completion-in-nlp/ (дата обращения: 10.02.2024 г.).

14. Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. // Journal of machine learning research. 2019. – 67 с.

15. Mohamed El-Geish. Gestalt: a Stacking Ensemble for SQuAD2.0. // Stanford CS224N Natural Language Processing with Deep Learning. 2020. – 11 с.

16. Привет! Меня зовут НОНА. [Электронный ресурс] URL: https://oboi-palitra.ru/about_company/news/zdravstvuyte-menya-zovut-nona-u-fabriki-palitra-poyavilsya-svoy-chat-bot/ (дата обращения: 10.02.2024 г.).

17. Привет, я робот Макс! Как устроен цифровой ассистент Госуслуг. [Электронный ресурс] URL: https://habr.com/ru/companies/rtlabs/articles/587034/ (дата обращения: 10.02.2024 г.).

18. Как в Альфа-Банке создали чат-бот, который понимает и отвечает, словно человек. [Электронный ресурс] URL: https://vc.ru/alfabank/581645-kak-v-alfa-banke-sozdali-chat-bot-kotoryy-ponimaet-i-otvechaet-slovno-chelovek (дата обращения: 10.02.2024 г.).

19. Альфа-Помощник стал лучшим чат-ботом в России. [Электронный ресурс] URL: https://alfabank.ru/news/t/release/alfa-pomoshchnik-stal-luchshim-chat-botom-v-rossii/ (дата обращения: 10.02.2024 г.).

20. WTF Is a Vector Database: A Beginners Guide! [Электронный ресурс] URL: https://dev.to/pavanbelagatti/wtf-is-a-vector-database-a-beginners-guide-16p (дата обращения: 29.04.2024 г.).

21. Chroma. [Электронный ресурс] URL: https://docs.trychroma.com/ (дата обращения: 10.05.2024 г.).

22. PyCharm. The Python IDE for data science and web development. [Электронный ресурс] URL: https://www.jetbrains.com/pycharm/ (дата обращения: 10.05.2024 г.).

23. Библиотека pyTelegramBotAPI. [Электронный ресурс] URL: https://pytba.readthedocs.io/ru/latest/index.html (дата обращения: 29.04.2024 г.).

24. HuggingFace. [Электронный ресурс] URL: https://huggingface.co/ (дата обращения: 10.05.2024 г.).

25. Excel. [Электронный ресурс] URL: https://www.microsoft.com/ru-ru/microsoft-365/excel (дата обращения: 12.05.2024 г.).

26. FRED-T5-LARGE_text_qa. [Электронный ресурс] URL: https://huggingface.co/Den4ikAI/FRED-T5-LARGE_text_qa (дата обращения: 15.05.2024 г.).

27. Automated metrics for evaluating the quality of text generation. [Электронный ресурс] URL: https://blog.paperspace.com/automated-metrics-for-evaluating-generated-text/ (дата обращения: 01.05.2024 г.).

28. Metric: bleu. [Электронный ресурс] URL: https://huggingface.co/spaces/evaluate-metric/bleu (дата обращения: 01.02.2024 г.).

29. Metric: rouge. [Электронный ресурс] URL: https://huggingface.co/spaces/evaluate-metric/rouge (дата обращения: 01.05.2024 г.).

30. Модульное тестирование от А до Я. [Электронный ресурс] URL: https://otus.ru/journal/modulnoe-testirovanie-ot-a-do-ya/ (дата обращения: 20.05.2024 г.).

ПРИЛОЖЕНИЯ

Приложение А. Спецификация вариантов использования

Спецификация вариантов использования (ВИ) системы приведена в таблицах 1–2.

Таблица 1 – Спецификация прецедента «Получить ответ на вопрос»

Прецедент: Получить ответ на вопрос
ID: 1
Аннотация: Получение ответа на вопрос в виде текстового сообщения в мессенджере
Главные актеры: Пользователь
Второстепенные актеры: Отсутствует
Предусловия: Чат-бот запущен
Основной поток: Пользователь отправляет вопрос в виде текстового сообщения
Постусловия: Система отправляет пользователю ответ в виде текстового сообщения
Альтернативные потоки: 2

Таблица 2 – Спецификация альтернативного потока «Обработка нерелевантного вопроса»

Альтернативный поток: Обработка нерелевантных вопросов
ID: 2
Аннотация: Система просит пользователя задать вопрос корректным образом
Главные актеры: Пользователь
Второстепенные актеры: Отсутствует
Предусловия: Пользователь отправил нерелевантный вопрос
Альтернативные потоки: Система отправляет сообщение о некорректности заданного вопрос и предлагает пользователю задать вопрос более корректным образом
Постусловия: Нет

Приложение Б. Программные коды системы

Листинг 1 – Полный список используемых библиотек

```
absl-py==2.1.0
aiohttp==3.9.3
aiosignal==1.3.1
annotated-types==0.6.0
anyio==4.3.0
asgiref==3.8.1
async-timeout==4.0.3
attrs==23.2.0
backoff==2.2.1
bcrypt==4.1.2
bleu==0.3
breadability==0.1.20
build==1.2.1
cachetools==5.3.3
certifi==2024.2.2
chardet==5.2.0
charset-normalizer==3.3.2
chroma-hnswlib==0.7.3
chromadb==0.4.24
click==8.1.7
colorama==0.4.6
coloredlogs==15.0.1
dataclasses-json==0.6.4
datasets==2.19.1
Deprecated==1.2.14
dill==0.3.8
docopt==0.6.2
efficiency==2.0
evaluate==0.4.2
exceptiongroup==1.2.0
fastapi==0.110.1
filelock==3.13.3
flatbuffers==24.3.25
frozenlist==1.4.1
fsspec==2024.3.1
gensim==4.3.2

googleapis-common-protos==1.63.0
greenlet==3.0.3
grpcio==1.62.1
h11==0.14.0
httptools==0.6.1
huggingface-hub==0.22.2
humanfriendly==10.0
idna==3.6
importlib-metadata==7.0.0
importlib_resources==6.4.0
Jinja2==3.1.3
joblib==1.3.2
jsonpatch==1.33
jsonpointer==2.4
kubernetes==29.0.0
langchain==0.1.14
langchain-community==0.0.31
langchain-core==0.1.40
langchain-text-splitters==0.0.1
langsmith==0.1.40
lxml==5.2.2
markdown-it-py==3.0.0
```

```
MarkupSafe==2.1.5
marshmallow==3.21.1
mdurl==0.1.2
mmh3==4.1.0
monotonic==1.6
mpmath==1.3.0
multidict==6.0.5
multiprocess==0.70.16
mypy-extensions==1.0.0
networkx==3.2.1
nltk==3.8.1
numpy==1.26.4
oauthlib==3.2.2
onnxruntime==1.17.1
opentelemetry-api==1.24.0
opentelemetry-exporter-otlp-proto-common==1.24.0
opentelemetry-exporter-otlp-proto-grpc==1.24.0
opentelemetry-instrumentation==0.45b0
opentelemetry-instrumentation-asgi==0.45b0
opentelemetry-instrumentation-fastapi==0.45b0
opentelemetry-proto==1.24.0
opentelemetry-sdk==1.24.0
opentelemetry-semantic-conventions==0.45b0
opentelemetry-util-http==0.45b0
orjson==3.10.0
overrides==7.7.0
packaging==23.2
pandas==2.2.2
posthog==3.5.0
protobuf==4.25.3
pulsar-client==3.4.0
pyarrow==16.0.0
pyarrow-hotfix==0.6
pyasn1==0.6.0
pyasn1_modules==0.4.0
pycountry==23.12.11
pydantic==2.6.4
pydantic_core==2.16.3
Pygments==2.17.2
PyPika==0.48.9
pyproject_hooks==1.0.0
pyreadline3==3.4.1
pyTelegramBotAPI==4.16.1
python-dateutil==2.9.0.post0
python-dotenv==1.0.1
pytz==2024.1
PyYAML==6.0.1
regex==2023.12.25
requests==2.31.0
requests-oauthlib==2.0.0
rich==13.7.1
rouge-metric==1.0.1
rouge_score==0.1.2
rsa==4.9
safetensors==0.4.2
scikit-learn==1.4.1.post1
scipy==1.10.1
sentencepiece==0.2.0
shellingham==1.5.4
six==1.16.0
smart-open==7.0.4
```

```
sniffio==1.3.1
SQLAlchemy==2.0.29
starlette==0.37.2
stop-words==2018.7.23
sumy==0.11.0
sympy==1.12
telebot==0.0.5
tenacity==8.2.3
threadpoolctl==3.4.0
tokenizers==0.15.2
tomli==2.0.1
torch==2.2.2
tqdm==4.66.2
transformers==4.39.3
typer==0.12.1
typing-inspect==0.9.0
typing_extensions==4.10.0
tzdata==2024.1
urllib3==2.2.1
uvicorn==0.29.0
watchfiles==0.21.0
websocket-client==1.7.0
websockets==12.0
wrapt==1.16.0
xxhash==3.4.1
yarl==1.9.4
zipp==3.18.1
```

Листинг 2 – Обработчик векторной базы данных `ChromaHandler`

```
class ChromaHandler:
    def __init__(self, model: ModelHandler):
        self.model = model
        self.chroma_client = chromadb.PersistentClient(path=CHROMA_PATH)

    @staticmethod
    def _load_documents() -> list[str]:
        """
        Load all documents from dataset
        :return: list of documents
        """
        df = pd.read_csv(DATA_PATH, delimiter=';', index_col=False)
        return [clean_text(f"{row['question']} {row['answer']}") for _, row
in df.iterrows()]

    def save_to_chroma_db(self):
        """
        Save all documents to Croma database
        :return: None
        """
        collection = self.chroma_client.get_or_create_collection(
            name=CHROMA_NAME,
            metadata={"hnsw:space": "cosine"},
            embedding_function=ChromaEmbeddingFunction(),
        )
        documents = self._load_documents()
        document_ids = list(map(lambda x: f"id{x[0]}",
enumerate(documents)))

        embeddings = list(map(self.model.get_embeddings, documents))
```

```
        batch_size = 165
        for idx in range(0, len(document_ids), batch_size):
            collection.add(
                ids=document_ids[idx: idx + batch_size],
                documents=documents[idx: idx + batch_size],
                embeddings=embeddings[idx: idx + batch_size])

    def get_close_texts(self, query_text: str) -> list[str]:
        """
        Get all closest documents from Croma by query text
        :param query_text: query text
        :return: list of closest documents        """
        collection = self.chroma_client.get_collection(name=CHROMA_NAME)
        response_text = collection.query(
            query_embeddings=self.model.get_embeddings(query_text),
            n_results=7)
        prompt_texts = [text for text in response_text['documents'][0]
                        if is_similar_texts(query_text, text, self.model)]
        return prompt_texts
```

Листинг 3 – Проверка точности ответов

```
import os.path
import pandas as pd
import evaluate

from src.db import ChromaHandler
from src.model_handler import ModelHandler
from src.text_process import remove_model_tags
from src.text_process import remove_not_ascii
from tests import TESTS_SOURCE_DIR

class QualityComparator:
    def __init__(self):
        self.model = ModelHandler()
        self.db = ChromaHandler(self.model)
        self.rouge = evaluate.load("rouge")
        self.bleu = evaluate.load("bleu")
        df = pd.read_csv(os.path.join(TESTS_SOURCE_DIR, "tests.csv"),
delimiter=';', index_col=False)
        self.test_questions = df["question"]
        self.test_answers = df["answer"]


    def __get_prediction(self, question):
        """

        :param question:
        :return:
        """
        similar_texts = self.db.get_close_texts(question)
        context = ''.join(similar_texts)
        prompt = '''<SC6>Текст: {}\nВопрос: {}\nОтвет: <extra_id_0>
                 '''.format(context, question)
        answer = remove_model_tags(self.model.generate(prompt))
        return answer

    def __calculate_metrics(self):
        preds = list()
        references = self.test_answers
```

```
        for count, question in enumerate(self.test_questions):
            preds.append(self.__get_prediction(remove_not_ascii(question)))
            print(f'{count + 1}/{len(self.test_questions)}')

        results = self.rouge.compute(predictions=preds,
                                     references=references)
        results['bleu'] = self.bleu.compute(predictions=preds,
                                            references=references)['bleu']

        return results

    def display_results(self) -> None:
        results = self.__calculate_metrics()
        print(f'bleu: {results["bleu"]} | rouge1: {results["rouge1"]} |
rouge2 {results["rouge2"]} | '
              f'rougeL {results["rougeL"]} | rougeLsum
{results["rougeLsum"]}')

if __name__ == '__main__':
    qc = QualityComparator()
    qc.display_results()
```

Приложение В. Тестирование системы

Листинг 4 – Unit-тестирование

```
import unittest
from unittest.mock import MagicMock

import numpy as np

from src.text_process import (
    remove_not_ascii,
    remove_invalid_chars,
    clean_text,
    text_similarity,
    is_similar_texts,
    get_summary,
    remove_model_tags
)


class TextProcessTest(unittest.TestCase):
    def test_remove_not_ascii(self):
        self.assertEqual(remove_not_ascii('Hello, мир!'), 'Hello, мир!')
        self.assertEqual(remove_not_ascii('Hello, world! 123'), 'Hello,
world! 123')
        self.assertEqual(remove_not_ascii('Поддерживается кириллица! 🚀'),
'Поддерживается кириллица! ')
        self.assertEqual(remove_not_ascii('こんにちは'), '')

    def test_remove_invalid_chars(self):
        self.assertEqual(remove_invalid_chars('Hello‘ “world”!  '), "Hello'
\"world\"! ")
        self.assertEqual(remove_invalid_chars("‘Test‘ “case” "), "'Test'
\"case\" ")
        self.assertEqual(remove_invalid_chars("Normal text without special
chars"), "Normal text without special chars")

    def test_clean_text(self):
        self.assertEqual(clean_text('Hello‘ “world”! Привет, мир!'),
"Hello' \"world\"! Привет, мир!")
        self.assertEqual(clean_text('Invalid ‘symbols‘ and non-ascii こんに
ちは'), "Invalid 'symbols' and non-ascii ")

    def test_text_similarity(self):
        model = MagicMock()
        model.get_embeddings = MagicMock(side_effect=[
            np.array([1, 2, 3]),
            np.array([1, 2, 3])
        ])
        question = "What is AI?"
        text = "Artificial Intelligence"
        self.assertAlmostEqual(text_similarity(question, text, model), 1.0)

    def test_is_similar_texts(self):
        model = MagicMock()
        model.get_embeddings = MagicMock(side_effect=[
            np.array([1, 2, 3]),
            np.array([1, 2, 3])
        ])
        self.assertTrue(is_similar_texts("What is AI?", "Artificial
Intelligence", model))
        model.get_embeddings = MagicMock(side_effect=[
            np.array([1, 2, 3]),
```

```
            np.array([-1, -2, -3])
        ])
        self.assertFalse(is_similar_texts("What is AI?", "Quantum
Computing", model))

    def test_get_summary(self):
        text = "This is a test text. It contains several sentences. The
purpose is to summarize this text."
        summary = get_summary(text)
        self.assertTrue(len(summary) < len(text))

    def test_remove_model_tags(self):
        self.assertEqual(remove_model_tags("This is a <tag>test</tag>."),
"This is a test.")
        self.assertEqual(remove_model_tags("<br>Remove <b>all</b> tags."),
"Remove all tags.")
        self.assertEqual(remove_model_tags("Text without tags."), "Text
without tags.")
```

Приложение Г. Процесс регистрации чат-бота

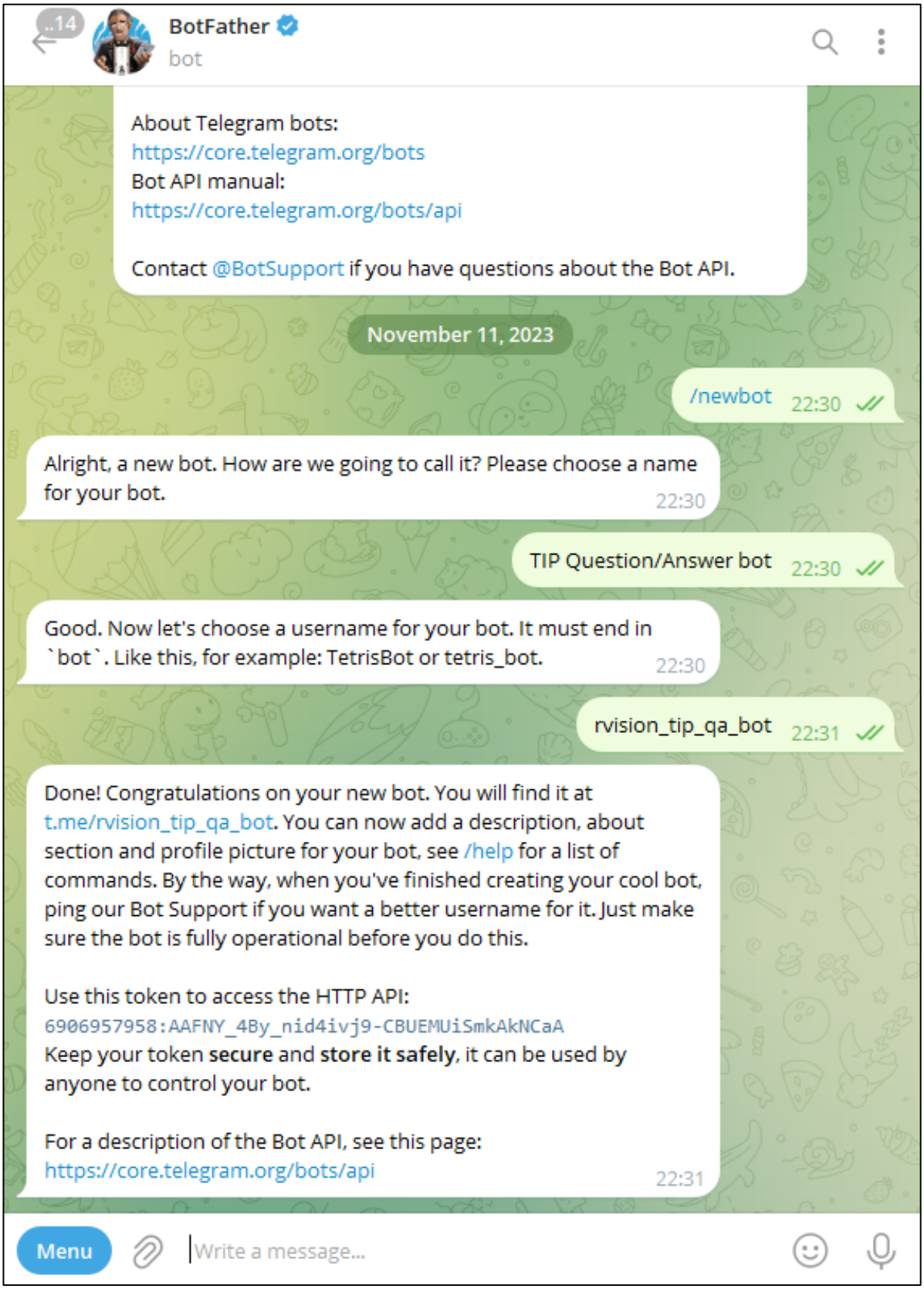


Рисунок 1 – Регистрация чат-бота

Приложение Д. Диаграмма деятельности

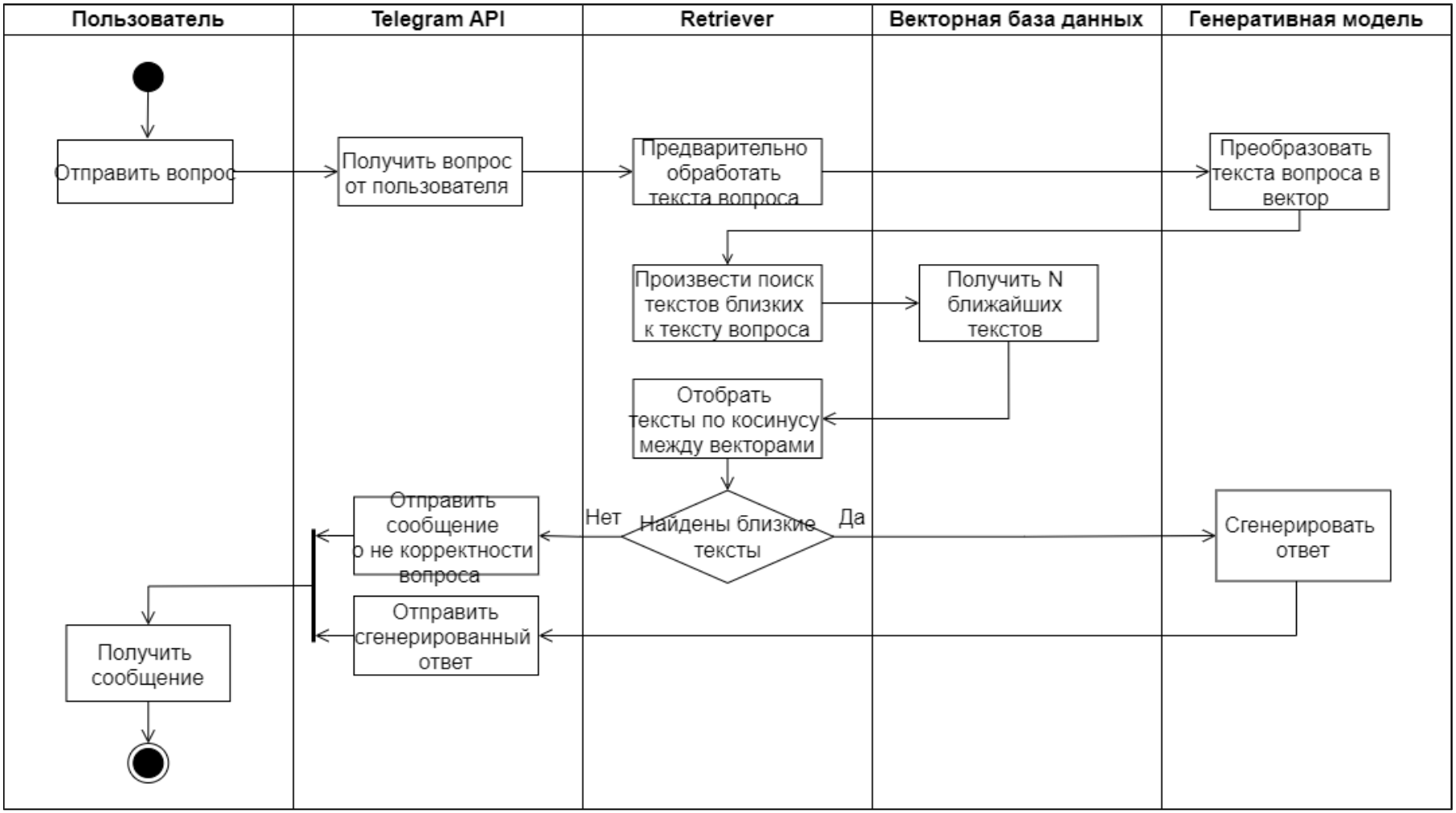


Рисунок 2 – Диаграмма деятельности