МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ   
РОССИЙСКОЙ ФЕДЕРАЦИИ  
Федеральное государственное автономное образовательное учреждение высшего образования  
"ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ"  
ШКОЛА КОМПЬЮТЕРНЫХ НАУК  
КАФЕДРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

ОТЧЁТ  
О РАЗРАБОТКЕ ПРОЕКТА ПО ДИСЦИПЛИНЕ  
«ИНТЕЛЛЕКТУАЛЬНЫЕ СИСТЕМЫ ПОДДЕРЖКИ ПРИНЯТИЯ РЕШЕНИЙ»  
на тему: Разработка системы автоматизированной суммаризации текста документов МФЦ на основе графов знаний

Выполнили студенты 4 курса МОиАИС:  
Жменько Артём Юрьевич  
Усачёва Елизавета Юрьевна

Руководитель проекта:  
Заведующий кафедры программного обеспечения  
Воробьёва Марина Сергеевна

**Оглавление**

[**ВВЕДЕНИЕ 3**](#_heading=h.rh3pib980z96)

[**ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ 5**](#_heading=h.rlajz62ichmb)

[1.1 Анализ существующих проблем 5](#_heading=h.u01sz81aqi4g)

[1.2 Обзор существующих решений 5](#_heading=h.m0eag593c87p)

[1.3 Подход к решению 6](#_heading=h.6f9oqvic3zr5)

[1.4 Целевая аудитория 6](#_heading=h.mvpo7qdffolk)

[1.5 Ценность предлагаемого решения 7](#_heading=h.fxqg34u1kk8m)

[**ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА 8**](#_heading=h.l66gbx3jgac7)

[2.1 Постановка задачи 8](#_heading=h.w0szz7ldxcmv)

[2.2 Используемые технологии 8](#_heading=h.on0pwl5mai6q)

[2.3 Описание информационного обеспечения 10](#_heading=h.117nveek0vxb)

[2.4 Разработка приложения 10](#_heading=h.gfx5rwknfeaz)

[2.4.1 Архитектура приложения 11](#_heading=h.q661mrbvwnf)

[2.4.2 Дообучение модели 12](#_heading=h.levho54plbpx)

[2.5 Интерфейс пользователя 12](#_heading=h.soeu6j4zofvm)

[2.6 Тестирование системы 13](#_heading=h.evr2xpnhsz80)

[2.6.1 Методы тестирования 13](#_heading=h.duap8q2ryckd)

[2.6.2 Результаты тестирования 13](#_heading=h.e0as6dmwg5r8)

[2.6.3 Выявленные проблемы и их решения 14](#_heading=h.ajnx76mvcsy9)

[**ГЛАВА 3. ОРГАНИЗАЦИЯ РАБОТЫ 15**](#_heading=h.qhickro6nxu5)

[3.1. Взаимодействие с заказчиком 15](#_heading=h.6j2nlttx1kw7)

[3.2. Распределение задач внутри команды 15](#_heading=h.ozms04dmz79v)

[3.3. Результаты организации работы 16](#_heading=h.1hp1tl8kp254)

[**ЗАКЛЮЧЕНИЕ 17**](#_heading=h.5w8smyvux0pc)

[**СПИСОК ЛИТЕРАТУРЫ 19**](#_heading=h.2fwii4rkj6s2)

[ПРИЛОЖЕНИЕ 21](#_heading=h.xcchotz6xr4b)

# ВВЕДЕНИЕ

Современные многофункциональные центры (МФЦ) предоставляют широкий спектр государственных и муниципальных услуг, охватывающих различные аспекты повседневной жизни граждан. Однако несмотря на попытки сделать получение услуг удобным, процесс взаимодействия с МФЦ для многих граждан остаётся сложным. Одной из ключевых проблем является обилие официальной документации, сопровождающей процесс оказания услуг. Клиентам предоставляются объёмные документы с подробным описанием порядка получения услуги, перечня необходимых документов, возможных исключений и частных случаев.

Чрезмерное использование формального языка, сложных терминов и громоздких формулировок затрудняет восприятие информации, особенно для граждан, не обладающих юридическими или техническими знаниями [1, 11]. В результате клиенты сталкиваются с трудностями при подготовке необходимых документов и часто приходят в МФЦ с неполным или неверным пакетом. Это приводит к повторным посещениям, тратам времени и ресурсов, а также к росту неудовлетворённости пользователей услугами [2].

Актуальность данной проблемы усиливается с увеличением потока клиентов и разнообразия предоставляемых услуг. Для решения обозначенных проблем необходимо внедрение автоматизированных систем, которые бы помогали гражданам быстро и легко ориентироваться в сложной документации, извлекая из неё только ключевые сведения.

Настоящий проект направлен на разработку системы для автоматической обработки текстов документов МФЦ – “МФЦ Простые документы”. Его задача – выделить основные требования для получения услуги из содержания документа и представить их в удобной для пользователя форме. Предлагаемая система поможет сократить время на ознакомление с информацией, минимизировать ошибки и увеличить удовлетворённость клиентов.

# ГЛАВА 1. ОБЗОР ПРЕДМЕТНОЙ ОБЛАСТИ

## 1.1 Анализ существующих проблем

Современные МФЦ выполняют важную роль в предоставлении государственных и муниципальных услуг. Однако сложность взаимодействия с МФЦ зачастую обуславливается обилием документации, которая носит формальный характер. Это приводит к затруднениям в понимании ключевой информации, необходимой для получения услуги, особенно для граждан, не обладающих достаточными юридическими или административными знаниями [1].

Результатом этого становятся:

* Повторные визиты в МФЦ из-за ошибок в подготовке документов.
* Повышение нагрузки на сотрудников центра.
* Рост неудовлетворённости клиентов, связанный с излишне сложными процессами [2].

## 1.2 Обзор существующих решений

На данный момент существует ряд технологий и инструментов, которые частично решают проблемы работы с текстовыми данными:

1. Системы управления документами (DMS): предоставляют средства для организации и хранения документов, но не адаптируют их под запросы пользователей [4].
2. Сервисы упрощения текста: такие как Grammarly и Hemmingway App, помогают улучшить читаемость текста, но не рассчитаны на обработку официальных документов [12, 13].
3. Платформы для графов знаний: Ontotext и Stardog предоставляют сложные решения для анализа и работы с графами данных, но их интеграция требует значительных усилий [12, 15].
4. Модели NLP: такие как Sentence-BERT и GPT, демонстрируют высокую точность в обработке текста, но требуют дополнительной настройки для работы с узкоспециализированными данными [7, 8].

Такие решения либо недостаточно специализированы, либо требуют значительных ресурсов для настройки и интеграции.

## 1.3 Подход к решению

Для решения проблемы наш проект применяет следующие современные технологии:

1. Модель Sentence-BERT для поиска семантически релевантной информации в текстах документов [7].
2. Генеративные модели (GPT) для перефразирования текста, адаптации его к запросам пользователей [8].
3. Retrieval-Augmented Generation (RAG): методология, совмещающая поиск релевантных данных и генерацию ответа, что позволяет обрабатывать текстовые запросы с меньшими галлюцинациями модели [5].
4. База данных

Перечисленные подходы позволяют создать систему, которая обеспечивает:

* Точное извлечение информации, релевантной запросу пользователя.
* Упрощение текста для улучшения восприятия данных.
* Автоматизацию взаимодействия с документацией, что экономит время клиентов.

## 1.4 Целевая аудитория

Основными пользователями системы являются клиенты МФЦ, которым требуется:

* Быстрый доступ к ключевой информации.
* Снижение времени на подготовку документов и получения услуги.

Система также снижает нагрузку на сотрудников МФЦ, позволяя им больше времени уделять нестандартным случаям и сложным вопросам [1].

## 1.5 Ценность предлагаемого решения

Внедрение автоматизированной системы анализа и аннотирования текста предоставляет значимые преимущества:

* Уменьшение времени, затрачиваемого клиентами на изучение документации.
* Снижение вероятности ошибок при подготовке документов.
* Повышение качества обслуживания за счёт доступности и удобства представленния информации [11].

Таким образом, разрабатываемая система направлена на решение ключевых проблем взаимодействия граждан с МФЦ, обеспечивая современный и удобный инструмент для работы с документами.

# ГЛАВА 2. РЕАЛИЗАЦИЯ ПРОЕКТА

## 2.1 Постановка задачи

Основной задачей проекта является создание автоматизированной системы обработки текстов документов, предоставляемых клиентам МФЦ. Система должна позволять пользователю вводить запрос на естественном языке и получать ответ, сформированный на основе релевантных разделов документации.

Реализация проекта состоит из следующих этапов:

1. Предобработка исходного документа.
2. Настройка модели для преобразования сегментов документа, которая находит в документах наиболее релевантные части текста для заданного запроса.
3. Настройка генеративной модели для преобразования найденных секций.
4. Создание пользовательского интерфейса в виде чат-бота в Telegram для удобного взаимодействия с системой.
5. Интеграция всех компонентов через API для обеспечения работоспособности системы.

## 2.2 Используемые технологии

1. Язык программирования Python

Описание: основной язык разработки проекта.

Возможности:

* Обширные библиотеки для обработки естественного языка (NLP).
* Простота интеграции с фреймворками и базами данных.
* Доступность использования моделей машинного обучения.

1. Фреймворк FastAPI

Описание: используется для реализации API.  
 Возможности:

* Быстрое создание RESTful API.
* Поддержка асинхронных операций для повышения производительности.

3. ElasticSearch

Описание: система для хранения и поиска текстовых эмбеддингов.  
 Возможности:

* Быстрый поиск похожих данных на основе векторных представлений.
* Масштабируемость и надёжность при работе с большими объёмами информации [10].

4. Sentence-BERT

Описание: модель для создания эмбеддингов текста.  
 Возможности:

* Определение семантической схожести между запросом и текстом с помощью метрики косинусного расстояния.
* Высокая точность при работе с текстами на естественном язык [7] .

5. GPT (Generative Pre-trained Transformer)

Описание: генеративная модель для упрощения и адаптации текста.  
 Возможности:

* Генерация генерация .
* Переформулирование сложных текстов для лучшего восприятия [8].

6. Telegram API

Описание: интерфейс для взаимодействия с пользователями через чат-бота.

Возможности:

* Реализация удобного пользовательского интерфейса.
* Поддержка текстовых и мультимедийных запросов [12].

## 2.3 Описание информационного обеспечения

Информационное обеспечение включает:

1. Входные данные: текстовые документы, предоставляемые МФЦ, полученные из Forcase API в формате JSON, названия услуг, описание услуг в виде документов, состоящих из иерархической структуры секций, конечным узлом которой является блок с текстом.
2. Обработка данных:

* Очистка лишних символов (ссылок,
* Векторизация текста с помощью Sentence-BERT.
* Хранение эмбеддингов в Elasticsearch для быстрого поиска.

1. Выходные данные: краткий и понятный текстовый ответ, сформированный на основе анализа документа и запроса пользователя.

## 2.4 Разработка приложения

Проект построен по модульному принципу

1. Модуль обработки запросов: анализирует входной запрос пользователя, создаёт эмбеддинги и ищет релевантные данные, отправляет данные в следующий модуль.
2. Модуль генерации текста: получает собранную информацию, формирует ответ на основе найденных данных и запроса пользователя.
3. API-модуль: обеспечивает связь между клиентом (Telegram-бот) и внутренними компонентами системы.
4. Интерфейсный модуль: чат-бот Telegram для взаимодействия с пользователем.

### 2.4.1 Архитектура приложения

Описание схемы архитектуры системы (см. рис. 1): пользователь взаимодействует с ботом. Бот отправляет номер документа и вопрос пользователя через Telegram API, загружается необходимый документ с соответствующим номером из Forcase API. Далее документ предобрабатывается (удаление лишних символов и деление на сегменты).

Происходит обращение к базе данных: Модель SBERT вычисляет семантические сходства между сегментами и вопросом пользователей: 5 наиболее подходящих и имеющих значение больше 0.3 отправляются в генерацию промпта.

Промпт собирается из инструкции и результатов работы модели, описанной ранее. Модель GPT на вход получает промпт, выводит ответ, который снова фильтрует модель SBERT. Полученный текст направляется через Telegram API пользователю в виде сообщения (см. приложение 2)

#### 

Рис. 1. Архитектура приложения

## 

## 2.4.2 Дообучение модели

Взятая за основу модель ruGPT3-large-based-on-gpt2 сильно галлюцинировала и не использовала заданный текст из промпта. Было решено сделать fine-tuning: дообучить модель на текстах документов МФЦ. Параметры для обучения следующие: 5 эпох по 27 шагов, шаг обучения: 5e-5, размер пакетов: 1. Дообучение заняло 6 часов. Необходимый результат был достигнут: модель начала учитывать контекст, заданный в промпте.

### 2.5 Интерфейс пользователя

Для взаимодействия с системой реализован Telegram-бот. Основные функции интерфейса:

* Приём текстовых запросов от пользователей.
* Обработка запросов с передачей данных на сервер.
* Выдача ответов пользователю в текстовой форме.
* Простота использования и минимальные требования к обучению.

Пример сценария работы:

1. Пользователь вводит запрос в Telegram-боте, например: «Какие документы нужны для оформления ипотеки?».
2. Бот анализирует запрос, извлекает релевантные данные из документов и генерирует ответ.
3. Ответ отправляется пользователю в удобной и понятной форме.

## 2.6 Тестирование системы

### 2.6.1 Методы тестирования

Для проверки корректности работы системы использовались следующие подходы:

1. Интеграционное тестирование компонентов: проверка взаимодействия между основными модулями системы: обработка запроса, поиск релевантных данных в Elasticsearch и генерация ответа.
2. Функциональное тестирование: оценка корректности ответов на основе тестового набора запросов, включающего типичные случаи и сложные формулировки, времени генерации ответов.
3. Измерение точности модели: применение метрик ROUGE и BERTScore (сравнение 20 ответов модели с ответами операторов МФЦ) для оценки качества генерации текста и поиска релевантной информации.

### 2.6.2 Результаты тестирования

1. Метрика BERTScore: достигнута средняя точность 0.75 при сопоставлении эмбеддингов текста из запросов и найденных секций: модель выдаёт ответы, семантически похожие на правильные.
2. Идентификация релевантных секций  
   Модель SBERT успешно выделила ключевые блоки текста в 85% тестовых запросов.
3. Среднее время генерации ответов ответов составило 24 секунды. Тестовый стенд: RTX 3060 Laptop, 32GB DDR4 3200MHz. Данный результат нельзя назвать идеальным, для сокращения времени ожидания необходимо увеличивать вычислительную мощность.

### 2.6.3 Выявленные проблемы и их решения

1. Галлюцинации модели: генерация информации, не относящейся к исходному запросу, затрудняла интерпретацию ответа.  
    Решение: применён метод Prompt-tuning, включающий изменение инструкции промпта и изменение гиперпараметров генерации текста.
2. Ограничения в обработке сложных формулировок запросов**:** модель иногда пропускала важные детали из сложных запросов.  
    Решение: модель дообучена на документах МФЦ, увеличено количество запросов с многозначными формулировками для улучшения fine-tuned модели.

Проведённое тестирование подтвердило, что система способна обрабатывать запросы пользователей, выделять релевантные данные из документов и предоставлять корректные ответы. Однако для повышения качества генерации ответов требуется дальнейшая оптимизация параметров модели и расширение тестового набора запросов.

# ГЛАВА 3. ОРГАНИЗАЦИЯ РАБОТЫ

## 3.1. Взаимодействие с заказчиком

В процессе выполнения проекта взаимодействие с заказчиком — представителями многофункционального центра (МФЦ) — было ограничено. Основные этапы взаимодействия включали:

1. Первичное согласование концепции проектаОбсуждение общей идеи и её согласования с целью определения ключевых аспектов системы: упрощения текстов документов и предоставления удобного интерфейса.
2. Определение требованийСбор информации о потребностях клиентов МФЦ, а также о наиболее частых проблемах, с которыми сталкиваются пользователи при изучении документов.
3. Промежуточные консультацииПроверка корректности выбранных подходов на начальных этапах разработки, включая подготовку пользовательского интерфейса и тестового сценария взаимодействия.

## 3.2. Распределение задач внутри команды

Работа над проектом была разделена между двумя участниками команды:

* Жменько Артём:
  1. Настройка языковых моделей для обработки текста.
  2. Интеграция технологий поиска релевантной информации (Elasticsearch) и генерации ответов (GPT-3).
  3. Тестирование и отладка системы.
* Усачёва Елизавета:
  1. Разработка интерфейса Telegram-бота.
  2. Настройка взаимодействия системы с API Telegram.
  3. Проведение тестирования и анализ результатов.

Эффективное распределение задач позволило одновременно работать над различными аспектами проекта, минимизируя задержки.

## 3.3. Результаты организации работы

Организация работы включала:

1. Определение целей и задач каждого этапаДля каждого этапа были установлены сроки выполнения и ожидаемые результаты. Это позволило команде сосредоточиться на ключевых аспектах проекта и завершить работу в установленный срок.
2. Еженедельное планирование и контроль прогрессаКаждую неделю обсуждались достигнутые результаты и определялись приоритеты на следующую неделю.
3. Результаты проектаВ результате проекта были разработаны:

* Система обработки запросов с использованием моделей NLP.
* Базовая версия Telegram-бота, позволяющего получать ответы на запросы пользователей.
* Тестирование системы с использованием ключевых метрик, таких как ROUGE и BERTScore.

Организация работы и распределение задач в команде позволили достичь поставленных целей и представить функциональную систему, которая решает ключевые проблемы, обозначенные в начале проекта.

# 

# ЗАКЛЮЧЕНИЕ

В ходе выполнения проекта "Разработка системы автоматизированной суммаризации текста документов МФЦ" были проанализированы основные проблемы взаимодействия граждан с многофункциональными центрами. Одной из ключевых трудностей, с которой сталкиваются пользователи, является сложность восприятия информации из официальных документов. Это обусловлено их громоздкой структурой, использованием формального языка и большим количеством деталей.

Для решения данной проблемы был разработан модуль автоматизированной обработки текстов, который:

* Определяет релевантные разделы документа, связанные с запросом пользователя.
* Генерирует краткий и понятный ответ с использованием языковых моделей.
* Предоставляет удобный пользовательский интерфейс в формате Telegram-бота.

Основные результаты проекта:

* Реализация модели поиска и обработки информации: настроена и интегрирована модель Sentence-BERT для создания эмбеддингов текста и поиска релевантных разделов документа.
* Реализована генерация ответов с использованием GPT-3.
* Разработка интерфейса системы: Telegram-бот позволяет пользователям взаимодействовать с системой, задавать запросы и получать ответы в удобной форме.
* Тестирование и оценка точности: проведено тестирование системы на тестовых запросах, что позволило выявить основные проблемы и оптимизировать модели.

Применение современных технологий

В проекте использованы передовые инструменты для обработки естественного языка (NLP) и работы с текстами, включая Elasticsearch и генеративные модели.

Выводы:

Разработанный прототип системы демонстрирует высокую актуальность и перспективность автоматизации процессов анализа текстовой информации в МФЦ.

Система позволяет упростить взаимодействие граждан с официальной документацией, минимизируя ошибки при подготовке документов.

Полученные результаты закладывают основу для дальнейшего развития проекта, включая:

Масштабирование системы для работы с различными категориями документов.

Оптимизацию модели для увеличения точности и производительности.

Добавление новых функций, таких как интеграция с базами данных МФЦ и расширение возможностей генерации ответов.

Проект достиг своих целей, предоставив рабочий инструмент для обработки запросов пользователей. Однако для полной реализации всех функциональных возможностей системы необходимо провести дополнительную оптимизацию и доработки. Полученные результаты доказывают, что использование технологий обработки естественного языка может значительно повысить удобство и эффективность взаимодействия граждан с многофункциональными центрами.

# 

# СПИСОК ЛИТЕРАТУРЫ

1. Молчанова И. И., Попова С. А. Разработка информационных систем на основе обработки естественного языка. — М.: Инфра-М, 2020. — 320 с.
2. Захарова Л. В., Карпов С. М. Введение в обработку естественного языка: теория и практическое применение. — СПб.: БХВ-Петербург, 2021. — 274 с.
3. Клименко И. В., Чернов П. С. Применение технологий искусственного интеллекта в государственном управлении. — М.: Наука, 2019. — 210 с.
4. Бондаренко О. И., Логинов А. В. Использование моделей машинного обучения в обработке текста. — СПб.: Питер, 2022. — 256 с.
5. Vaswani, A., Shazeer, N., Parmar, N., et al. Attention is all you need. *NeurIPS 2017*. — 2017.
6. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. Devlin, J., Chang, M. W., Lee, K., et al. — 2018.
7. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. Reimers, N., Gurevych, I. *arXiv:1908.10084*, 2019.
8. Radford, A., Narasimhan, K., Salimans, T., et al. Improving Language Understanding by Generative Pre-Training. OpenAI, 2018.
9. Hamid, S., M. GPT-3: Language Models are Few-Shot Learners. *arXiv preprint arXiv:2005.14165*. — 2020.
10. Elasticsearch: The Definitive Guide / Shay Banon, Clinton Gormley, and Zachary Tong. — O’Reilly Media, 2015. — 500 p.
11. Chomsky, N. Syntactic Structures. — The Hague: Mouton & Co, 1957.
12. Ontotext. (n.d.). [*https://ontotext.com*](https://ontotext.com) [Электронный ресурс].
13. Hemmingway App. [*https://www.hemingwayapp.com*](https://www.hemingwayapp.com) [Электронный ресурс].
14. Grammarly. [*https://www.grammarly.com*](https://www.grammarly.com) [Электронный ресурс].
15. Stardog. [*https://www.stardog.com*](https://www.stardog.com) [Электронный ресурс].
16. GitHub - OpenAI GPT-3: [*https://github.com/openai/gpt-3*](https://github.com/openai/gpt-3) [Электронный ресурс].
17. TensorFlow: Models and Examples. [*https://www.tensorflow.org/*](https://www.tensorflow.org/) [Электронный ресурс].

# ПРИЛОЖЕНИЕ

*Приложение 1. Примеры запросов и ответов системы.*

Вопрос: как вернуть госпошлину?

Ответ: Вы можете подать заявление о возврате в любом офисе ГАУ ТО "МФЦ" или в органе государственной власти в Управлении Росреестра по Тюменской области.

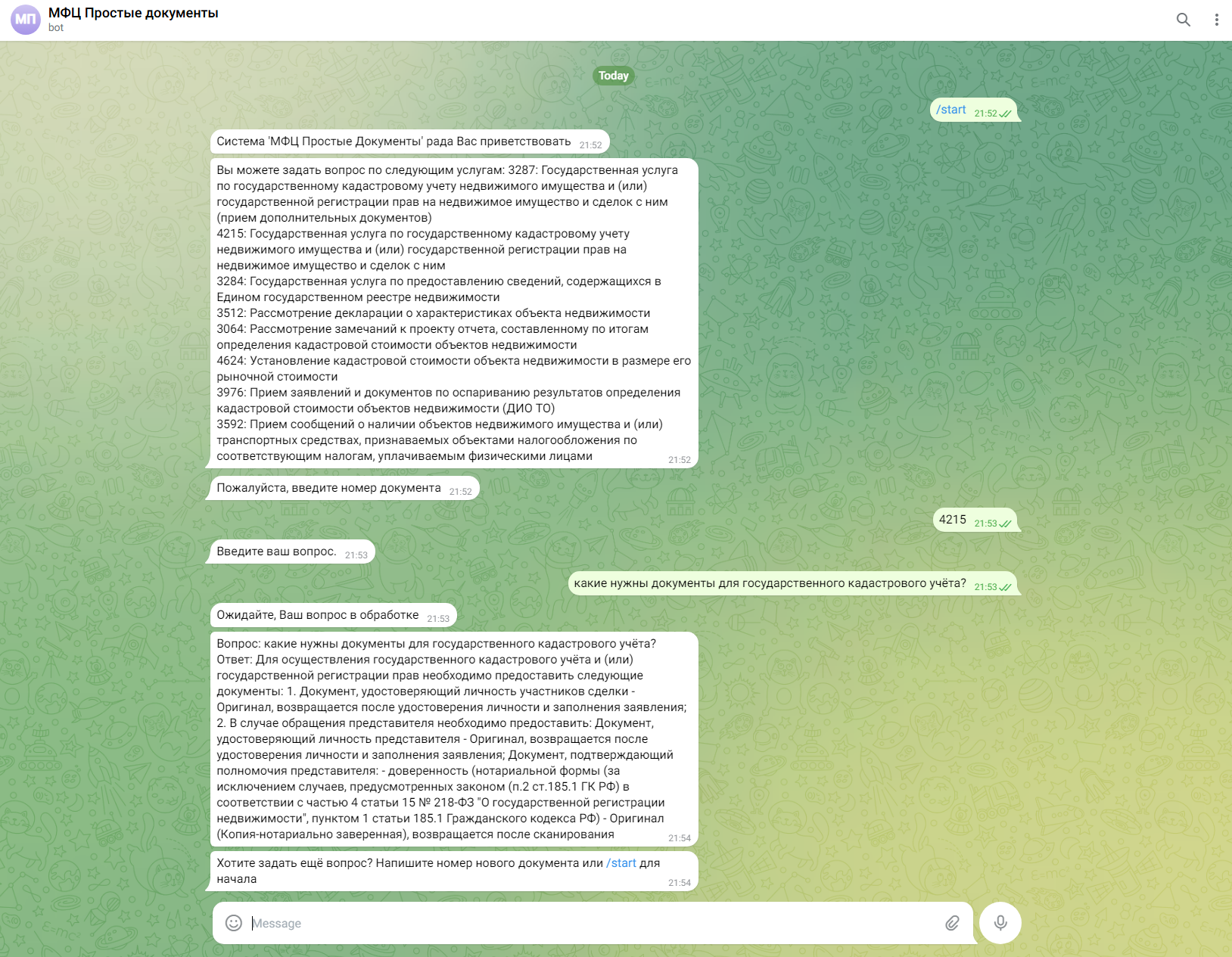
СРОКИ ВОЗВРАТА ГОСПОШЛИНЫ Возврат излишне уплаченной (взысканной) суммы государственной пошлины производится: - в любом офисе ГАУ ТО "МФЦ"; - в органе государственной власти в Управлении Росреестра по Тюменской области.

ПЕРЕЧЕНЬ ДОКУМЕНТОВ 1. Документ, удостоверяющий личность - Оригинал, возвращается после удостоверения личности; 2. Документ, подтверждающий полномочия представителя - Оригинал, возвращается после сканирования; 3. Документ, подтверждающий полномочия представителя юридического лица - Оригинал, возвращается после сканирования.

Вопрос: как зарегистрировать договор об ипотеке?

Ответ: Для регистрации договора об ипотеки необходимо обратиться в банк, в котором открыт счет заемщика. Также возможен результат в виде: уведомления об отказе, уведомления о возврате без рассмотрения, уведомления о прекращении.

*Приложение 2. Скриншоты интерфейса Telegram-бота.*



*Приложение 3. Исходный код проекта.*

from fastapi import FastAPI

from contextlib import asynccontextmanager

import threading

import time

import httpx

import re

from sentence\_transformers import SentenceTransformer

from sklearn.metrics.pairwise import cosine\_similarity

from transformers import GPT2LMHeadModel, GPT2Tokenizer

import torch

import json

TELEGRAM\_TOKEN = "7790465783:AAG8vsCeMB1ucgUNiEh0lTTisRVSbsZWsMk"

TELEGRAM\_API\_URL = f"https://api.telegram.org/bot{TELEGRAM\_TOKEN}"

user\_state = {}

class DocumentContainer:

prepared\_document = None

document\_embeddings = None

doc\_container = DocumentContainer()

device = torch.device("cuda" if torch.cuda.is\_available() else "cpu")

if device.type == 'cuda':

print('Для вычислений используются ядра cuda')

else:

print('Для вычислений используется процессор')

model\_name = 'ruGPT3-large/rugpt3\_finetuned'

retrieval\_model = SentenceTransformer("ai-forever/sbert\_large\_nlu\_ru", device=device.type)

tokenizer = GPT2Tokenizer.from\_pretrained(model\_name)

generating\_model = GPT2LMHeadModel.from\_pretrained(

model\_name,

do\_sample=True,

device\_map=device

)

service\_names\_path = 'data/doc\_links.json'

documents\_path = 'data/document\_text.json'

with open(service\_names\_path, encoding='utf-8') as file:

json\_content = file.read()

service\_blocks: list[dict] = json.loads(json\_content)

realty\_service\_names = dict()

for service\_names in service\_blocks:

for service\_name, service\_number in service\_names.items():

if 'недвижим' in service\_name.lower():

realty\_service\_names[service\_number] = service\_name

with open(documents\_path, encoding='utf-8') as file:

json\_documents = file.read()

document\_texts = json.loads(json\_documents)

def preprocess\_corpus(text):

text = text.replace("\n", " ").replace("\t", " ").strip()

text = re.sub(r"\s+", " ", text)

text = re.sub(r"●|•", "-", text)

text = re.sub(r"[«»]", "", text)

return text

def send\_message(chat\_id: int, text: str):

url = f"{TELEGRAM\_API\_URL}/sendMessage"

payload = {"chat\_id": chat\_id, "text": text}

httpx.post(url, json=payload)

def get\_updates(offset=None):

url = f"{TELEGRAM\_API\_URL}/getUpdates"

params = {"offset": offset, "timeout": 10}

response = httpx.get(url, params=params)

return response.json()

def process\_message(chat\_id, text):

if chat\_id not in user\_state:

user\_state[chat\_id] = {"step": "ask\_document"}

state = user\_state[chat\_id]

if state["step"] == "ask\_document":

send\_message(chat\_id, "Система 'МФЦ Простые Документы' рада Вас приветствовать")

send\_message(chat\_id, f"Вы можете задать вопрос по следующим услугам: "

f"{'\n'.join([f'{service\_number}: {service\_name}'

for service\_number, service\_name in realty\_service\_names.items()])}")

send\_message(chat\_id, "Пожалуйста, введите номер документа")

state["step"] = "get\_document"

elif state["step"] == "get\_document":

state["document\_number"] = text

if text.isnumeric():

send\_message(chat\_id, "Введите ваш вопрос.")

prepare\_document(int(text))

state["step"] = "get\_question"

else:

send\_message(chat\_id, "Введите номер документа корректно")

elif state["step"] == "get\_question":

send\_message(chat\_id, "Ожидайте, Ваш вопрос в обработке")

question = text

answer = get\_answer(question)

send\_message(chat\_id, answer)

send\_message(chat\_id, "Хотите задать ещё вопрос? Напишите номер нового документа или /start для начала")

state["step"] = "ask\_document"

def prepare\_document(document\_number):

selected\_document: list[str] = document\_texts[str(document\_number)]

prepared\_document: list[str] = []

is\_merge\_necessary = False

for line in selected\_document:

line = preprocess\_corpus(line)

if line.isupper():

prepared\_line: str = line[0] + line[1:].lower() + ': '

if is\_merge\_necessary:

prepared\_document[-1] += prepared\_line

else:

prepared\_document.append(prepared\_line)

is\_merge\_necessary = True

else:

if is\_merge\_necessary:

prepared\_document[-1] += line

is\_merge\_necessary = False

else:

prepared\_document.append(line)

doc\_container.prepared\_document = prepared\_document

doc\_container.document\_embeddings = retrieval\_model.encode(prepared\_document)

def get\_answer(question):

print('question embedding calculation')

query\_embedding = retrieval\_model.encode([question])

print('embeding similariry search')

similarities = cosine\_similarity(query\_embedding, doc\_container.document\_embeddings)

texts\_with\_similarities = tuple(zip(similarities[0], doc\_container.prepared\_document))

top\_texts = sorted(texts\_with\_similarities, key=lambda x: x[0], reverse=True)[:5]

top\_texts = [top\_text for top\_text in top\_texts if top\_text[0] > 0.3]

if len(top\_texts) == 0:

return 'Информация по вашему вопросу не была найдена в документе'

else:

for text in top\_texts:

print(text)

model\_input\_info = '\n'.join([text[1] for text in top\_texts])

query\_with\_context = (

f"<s>Инструкция: Отвечай кратко, для ответа используй только данные из контекста ниже:\n" +

f"{model\_input\_info}</s>\nВопрос: {question}\nОтвет:")

inputs = tokenizer.encode(query\_with\_context, return\_tensors="pt").to(device)

print('generating')

outputs = generating\_model.generate(

inputs,

num\_beams=3,

max\_new\_tokens=150,

early\_stopping=True,

temperature=0.2,

)

answer: str = tokenizer.decode(outputs[0], skip\_special\_tokens=False)

answer = answer[answer.find('</s>') + 5:]

print('ready')

return answer

def polling():

print("Запущен polling...")

offset = 0

while True:

try:

updates = get\_updates(offset)

if "result" in updates:

for update in updates["result"]:

offset = update["update\_id"] + 1

chat\_id = update["message"]["chat"]["id"]

text = update["message"].get("text", "")

process\_message(chat\_id, text)

except Exception as e:

pass

# print(f'Ну, всякое бывает: {e}')

time.sleep(1)

@asynccontextmanager

async def lifespan(app: FastAPI):

thread = threading.Thread(target=polling, daemon=True)

thread.start()

print("Polling успешно запущен!")

yield

print("Приложение останавливается...")

app = FastAPI(lifespan=lifespan)

@app.get("/")

def read\_root():

return {"message": "Telegram бот работает через polling и FastAPI!"}

import json

service\_names\_path = '../data/doc\_links.json'

documents\_path = '../data/document\_text.json'

# загрузка файла с услугами

with open(service\_names\_path, encoding='utf-8') as file:

json\_content = file.read()

service\_blocks: list[dict] = json.loads(json\_content)

realty\_service\_names = dict()

count = 0

for service\_names in service\_blocks:

for service\_name, service\_number in service\_names.items():

count += 1

if 'недвижим' in service\_name.lower():

realty\_service\_names[service\_number] = service\_name

realty\_service\_names

with open(documents\_path, encoding='utf-8') as file:

json\_documents = file.read()

document\_texts = json.loads(json\_documents)

document\_texts

document\_texts = {int(key): value for key, value in document\_texts.items()}

prepared\_texts = []

for key, value in realty\_service\_names.items():

if key in document\_texts:

# Добавляем в результат объединённое значение

prepared\_texts.append(value + '. ' + ' '.join(document\_texts[key]))

prepared\_texts

import re

def preprocess\_corpus(text):

# Убираем лишние пробелы и переносы строк

text = text.replace("\n", " ").replace("\t", " ").strip()

text = re.sub(r"\s+", " ", text) # Убираем лишние пробелы

# Заменяем маркеры списка

text = re.sub(r"●|•", "-", text)

# Убираем ненужные символы (если нужно)

text = re.sub(r"[«»]", "", text)

return text

training\_texts = [preprocess\_corpus(text) for text in prepared\_texts]

len(training\_texts[2])

from transformers import AutoTokenizer

from datasets import Dataset

def split\_into\_blocks(text: str, tokenizer, block\_size=1024):

tokens = tokenizer.encode(text)

return [tokens[i : i + block\_size] for i in range(0, len(tokens), block\_size)]

tokenizer = AutoTokenizer.from\_pretrained("ai-forever/rugpt3large\_based\_on\_gpt2")

training\_tokenized\_texts = []

for text in training\_texts:

training\_tokenized\_texts.extend(split\_into\_blocks(text, tokenizer))

print(len(training\_tokenized\_texts))

data = {"input\_ids": training\_tokenized\_texts}

dataset = Dataset.from\_dict(data)

dataset = dataset.map(lambda x: {"labels": x["input\_ids"]})

from transformers import AutoModelForCausalLM, Trainer, TrainingArguments

model\_name = "ai-forever/rugpt3large\_based\_on\_gpt2"

model = AutoModelForCausalLM.from\_pretrained(model\_name)

training\_args = TrainingArguments(

output\_dir="./results",

save\_strategy="steps",

save\_steps=54,

overwrite\_output\_dir=True,

per\_device\_train\_batch\_size=1,

per\_device\_eval\_batch\_size=1,

num\_train\_epochs=5,

logging\_dir="./logs",

logging\_steps=27,

learning\_rate=5e-5,

fp16=True

)

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=dataset,

tokenizer=tokenizer

)