**НИТУ «МИСИС»**

**Кафедра инженерной кибернетики**

**ОТЧЕТ**

по

**ЛАБОРАТОРНОЙ РАБОТЕ №4**

*«Разработка асинхронного чат-бота для мессенджера Telegram*

*с использованием языковых моделей архитектур GPT и LLaMA»*

учебная дисциплина

«Методы искусственного интеллекта»

**Студент: Осипова Елизавета Андреевна**

**Группа**: \_\_\_\_\_**БПМ-21-2**\_\_\_\_\_\_\_

**Преподаватель: Хонер П. Д.**

**Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Дата защиты: \_\_\_\_\_\_\_\_\_\_\_\_\_**

**2024 г.**

Оглавление

[**ВВЕДЕНИЕ** 3](#_Toc192445291)

[**1.** **Задание на лабораторную работу** 4](#_Toc192445292)

[**1.1. Основная цель работы** 4](#_Toc192445293)

[**1.2. Общие сведения** 4](#_Toc192445294)

[**1.3. Языковые модели, рекомендуемые к использованию** 4](#_Toc192445295)

[**2.** **Выполнение лабораторной работы** 5](#_Toc192445296)

[**2.1. Выбор библиотеки для создания чат-бота** 5](#_Toc192445297)

[**2.2. Выбор предметной области** 5](#_Toc192445298)

[**2.3. Проектирование чат-бота** 5](#_Toc192445299)

[**2.4. Разработка асинхронного чат-бота** 5](#_Toc192445300)

[**2.5. Использованные технологии и библиотеки** 6](#_Toc192445301)

[**2.6. Сравнительный анализ языковых моделей** 6](#_Toc192445302)

[**2.7. Проблемы и ограничения** 7](#_Toc192445303)

[**2.8. Модулярность кода** 7](#_Toc192445304)

[**2.9. Использование файла `educational\_keywords.txt`** 8](#_Toc192445305)

[**3.** **Структура проекта** 11](#_Toc192445306)

[**3.1.** **Модуль tbot.py** 11](#_Toc192445307)

[**3.2.** **Модуль handlers.py** 12](#_Toc192445308)

[**3.3.** **Модуль models.py** 17](#_Toc192445309)

[**3.4.** **Модуль utils.py** 19](#_Toc192445310)

[**ВЫВОДЫ** 23](#_Toc192445311)

[**ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА** 24](#_Toc192445312)

[**ПРИЛОЖЕНИЕ А. Работа чат-бота** 25](#_Toc192445313)

[**ПРИЛОЖЕНИЕ Б. Код чат-бота** 28](#_Toc192445314)

# **ВВЕДЕНИЕ**

Целью данной лабораторной работы является разработка асинхронного чат-бота для мессенджера Telegram, который использует языковые модели архитектур GPT и LLaMA для генерации ответов на вопросы пользователей. Чат-бот предназначен для помощи в учебных вопросах, таких как предоставление расписания, напоминаний о дедлайнах и ответов на образовательные запросы. В процессе выполнения работы были изучены и применены технологии асинхронного программирования, а также проведен сравнительный анализ языковых моделей rugpt3small\_based\_on\_gpt2 и Llama-3.2-3B-Instruct.

**Что такое асинхронный бот и чем он отличается от синхронного?**

Асинхронный и синхронный боты различаются по способу обработки запросов и взаимодействия с пользователем.

*Синхронный бот:*

- Синхронный бот обрабатывает запросы последовательно. То есть он выполняет один запрос за раз и не может начать обработку следующего запроса, пока не завершит предыдущий.

- Пользователь должен дождаться завершения операции, прежде чем отправить следующий запрос. Это может привести к задержкам, особенно если запрос требует длительной обработки.

- Синхронные боты обычно проще в разработке и отладке.

*Асинхронный бот:*

- Асинхронный бот может обрабатывать несколько запросов одновременно. Он не блокируется на одном запросе и может переключаться между задачами по мере готовности данных или ресурсов.

- Пользователи могут отправлять несколько запросов подряд без ожидания завершения предыдущего.

- Асинхронные боты более эффективно используют вычислительные ресурсы, так как они могут выполнять другие задачи, пока ждут ответа от внешнего сервиса или базы данных.

- Они более сложные в разработке и отладке.

*Как выбрать между асинхронным и синхронным ботами?*

- Синхронный бот может быть хорошим выбором для простых задач или в системах, где запросы обрабатываются быстро и последовательно.

- Асинхронный бот подходит для более сложных систем, где требуется высокая производительность и возможность обрабатывать множество запросов одновременно, особенно в средах с высокой нагрузкой.

# **Задание на лабораторную работу**

## **1.1. Основная цель работы**

Основная цель работы — разработать асинхронный чат-бот для Telegram, который использует языковые модели rugpt3small\_based\_on\_gpt2 и Llama-3.2-3B-Instruct для генерации ответов на вопросы пользователей. Чат-бот должен предоставлять функциональность, связанную с учебным процессом, такую как показ расписания, напоминаний и ответов на образовательные вопросы.

## **1.2. Общие сведения**

Чат-боты — это программы, которые имитируют общение с пользователем и помогают решать различные задачи. В данной работе чат-бот разработан для помощи в учебном процессе. Основное отличие синхронного и асинхронного подхода заключается в том, что асинхронный подход позволяет обрабатывать несколько запросов одновременно, что повышает производительность и реактивность бота.

## **1.3. Языковые модели, рекомендуемые к использованию**

В работе использовались следующие языковые модели:

1. *rugpt3small\_based\_on\_gpt2*

Модель ruGPT3small\_based\_on\_gpt2 является небольшой версией GPT-3, обученной на русском языке. Она основана на архитектуре GPT-2 и предназначена для генерации текстов на русском языке. Вот основные характеристики этой модели:

- Основана на GPT-2, что обеспечивает высокую производительность в задачах генерации текста.

- Специально обучена для работы с русским языком, что делает её более точной для обработки и генерации русскоязычных текстов.

- Это более компактная версия, которая может быть быстрее и менее требовательна к ресурсам по сравнению с более крупными моделями, такими как полные версии GPT-3.

- Используется для генерации текстов, ответов на вопросы, создания диалогов и других задач, связанных с обработкой естественного языка.

- Поддерживает параметры, такие как temperature для управления креативностью и max\_length для ограничения длины генерируемого текста.

Эта модель удобна для приложений, где требуется генерация текста на русском языке с ограниченными вычислительными ресурсами.

1. *Llama-3.2-3B-Instruct*

Модель Llama-3.2-3B-Instruct представляет собой модель семейства LLaMA (Large Language Model Meta AI), предназначенную для выполнения инструкций. Вот основные характеристики этой модели:

- Базируется на архитектуре трансформеров, которая хорошо подходит для задач обработки естественного языка.

- Содержит 3.2 миллиарда параметров, что делает её достаточно мощной для выполнения различных задач генерации и понимания текста.

- Специализирована на выполнении инструкций, что делает её полезной для задач, где требуется следовать конкретным указаниям или запросам пользователя.

- Подходит для генерации текстов, ответов на вопросы, создания контента и других задач, связанных с обработкой текста и взаимодействием на основе заданных инструкций.

- Поддерживает различные параметры управления генерацией текста, такие как temperature и max\_length, что позволяет настраивать креативность и длину ответа.

Эта модель полезна в приложениях, где требуется точное выполнение инструкций и генерация текстов высокого качества.

# **Выполнение лабораторной работы**

## **2.1. Выбор библиотеки для создания чат-бота**

Для разработки чат-бота была выбрана библиотека AsyncTeleBot, которая поддерживает асинхронное программирование и позволяет эффективно обрабатывать запросы пользователей.

## **2.2. Выбор предметной области**

Предметной областью чат-бота является учебный процесс. Чат-бот предоставляет функциональность для просмотра расписания, напоминаний о дедлайнах и ответов на образовательные вопросы.

## **2.3. Проектирование чат-бота**

Чат-бот имеет следующую функциональность:

- Показ расписания на текущий день.

- Показ списка дедлайнов.

- Ответы на образовательные вопросы с использованием языковых моделей rugpt3small\_based\_on\_gpt2 и Llama-3.2-3B-Instruct.

- Возможность выбора из двух моделей для генерации ответов.

## **2.4. Разработка асинхронного чат-бота**

Чат-бот разработан с использованием асинхронного подхода, что позволяет ему обрабатывать несколько запросов одновременно. Глубина вложенности функциональности составляет 5 уровней:

1. Главное меню.

2. Показ расписания.

3. Показ дедлайнов.

4. Выбор модели для генерации ответов.

5. Генерация ответов на образовательные вопросы.

## **2.5. Использованные технологии и библиотеки**

В процессе разработки чат-бота были использованы следующие технологии и библиотеки:

- AsyncTeleBot: Библиотека для создания асинхронного Telegram-бота.

- Hugging Face Transformers: Библиотека для работы с языковыми моделями GPT и LLaMA.

- Natasha: Библиотека для морфологического анализа русского языка.

- Dotenv: Для загрузки переменных окружения.

- Asyncio: Для реализации асинхронного программирования.

## **2.6. Архитектуры моделей**

### **2.6.1. Архитектура модели ruGPT3small\_based\_on\_gpt2**

Модель ruGPT3small\_based\_on\_gpt2 основана на архитектуре GPT-2, адаптированной для работы с русским языком.

### Архитектура GPT-2:

1. **Трансформер**:
   * Основывается на механизме внимания (attention), который позволяет модели учитывать контекст всего входного текста для генерации ответа.
   * Состоит из слоев энкодера, каждый из которых включает в себя многоголовочное внимание и позиционно-зависимые эмбеддинги.
2. **Многоголовочное внимание (Multi-Head Attention)**:
   * Позволяет модели одновременно учитывать разные части текста, улучшая понимание контекста.
3. **Фидфорвардные слои (Feedforward Layers)**:
   * Включают нелинейности и нормализацию, добавляя модели глубину и способность обучаться сложным зависимостям.
4. **Позиционные эмбеддинги**:
   * Обеспечивают модель информацией о порядке слов, так как трансформеры сами по себе не имеют встроенного понимания временной последовательности.
5. **Обучение на языковых данных**:
   * Модель обучена на большом корпусе русскоязычных текстов, что позволяет ей генерализовать и понимать различные контексты и стили языка.

### **2.6.2. Архитектура модели Llama-3.2-3B-Instruct**

Модель Llama-3.2-3B-Instruct является частью серии LLaMA (Large Language Model Meta AI), специально разработанной для выполнения инструкций.

### Архитектура LLaMA

1. **Трансформер**:
   * Как и другие современные языковые модели, LLaMA основана на архитектуре трансформеров, которая использует механизмы внимания для обработки и генерации текста.
2. **Многоголовочное внимание (Multi-Head Attention)**:
   * Позволяет модели учитывать различные части текста одновременно, улучшая понимание контекста и зависимостей между словами.
3. **Фидфорвардные слои (Feedforward Layers)**:
   * Обеспечивают нелинейные преобразования и нормализацию, позволяя модели обучаться сложным зависимостям в данных.
4. **Позиционные эмбеддинги**:
   * Включают информацию о порядке слов, что важно для понимания последовательности и структуры текста.
5. **Обучение на инструкциях**:
   * Специально обучена на корпусах данных, где важны точные инструкции и выполнение заданий, что делает её особенно полезной для задач, требующих следования указаниям.
6. **Оптимизация и эффективность**:
   * LLaMA разработана с акцентом на оптимизацию использования ресурсов, что позволяет ей эффективно работать при относительно большом количестве параметров.

## **2.7. Сравнительный анализ языковых моделей**

Сравним модели ruGPT3small\_based\_on\_gpt2 и Llama-3.2-3B-Instruct.

- Обе модели основаны на трансформерах, это обеспечивает эффективное внимание и обработку текста.

- ruGPT3small\_based\_on\_gpt2 использует архитектуру GPT-2, которая адаптированна для русского языка, а Llama-3.2-3B-Instruct базируется на LLaMA с акцентом на выполнение инструкций все зависимости от конкретного языка.

- ruGPT3small\_based\_on\_gpt2 более компактна, что делает её менее требовательной к ресурсам.

- Llama-3.2-3B-Instruct имеет 3.2 миллиарда параметров, это обеспечивает большую мощность и универсальность.

- ruGPT3small\_based\_on\_gpt2 оптимизирована для русского языка, тем самым она обеспечивая точность в русскоязычных приложениях, однако не во всех темах.

- Llama-3.2-3B-Instruct поддерживает много языков, включая английский, поэтому она считается подходящей для международных задач.

- Llama-3.2-3B-Instruct специализируется на выполнении инструкций, что делает её особенно полезной для задач, требующих четкого следования указаниям.

**Общие выводы:**

*- Основная задача:*

- Если основная задача — работа с русским языком и важна экономия ресурсов, ruGPT3small\_based\_on\_gpt2 будет неплохим выбором из-за её компактности и специализации на русском языке.

- Для задач, где требуется выполнение сложных инструкций и, например поддержка нескольких языков, Llama-3.2-3B-Instruct будет работать лучше благодаря своей мощности.

*- Ресурсы и масштабируемость:*

- ruGPT3small\_based\_on\_gpt2 подходит для приложений с ограниченными ресурсами, где важна только скорость ответов. В моем случае эта модель – не лучший вариант, она плохо отвечает на учебные вопросы.

- Llama-3.2-3B-Instruct лучше подходит для более сложных задач, где нет ограничения по вычислительным ресурсам и требуется высокая точность выполнения задач. В моем случае это лучший выбор.

Выбор модели зависит от требований к задаче и доступных вычислительных ресурсов.

## **2.8. Проблемы и ограничения**

Одной из основных проблем при использовании бесплатных моделей с Hugging Face является их низкая эффективность при обработке учебных запросов на русском языке. Только модель LLaMA Instruct показала удовлетворительные результаты. Для улучшения качества ответов требуется тонкая настройка (fine-tuning) моделей, что требует значительных вычислительных ресурсов.

## **2.9. Модулярность кода**

Код чат-бота был разработан с учетом принципов модулярности, что позволяет легко поддерживать и расширять функциональность бота. Каждый компонент системы выделен в отдельный модуль, что обеспечивает четкое разделение ответственности и упрощает тестирование и отладку. Основные модули, используемые в проекте:

1. handlers.py

Этот модуль содержит все обработчики сообщений и команд, которые взаимодействуют с пользователем. Каждая функция-обработчик отвечает за конкретную задачу, такую как показ расписания, напоминаний или генерация ответов на вопросы. Модуль также включает в себя логику для выбора модели и обработки учебных запросов.

2. models.py

В этом модуле реализована работа с языковыми моделями. Он отвечает за инициализацию моделей, их настройку и генерацию ответов на основе пользовательских запросов. Модуль использует библиотеку Hugging Face Transformers для взаимодействия с моделями GPT и LLaMA.

3. utils.py

Этот модуль содержит вспомогательные функции, такие как создание клавиатуры для меню, нормализация текста и проверка, является ли сообщение учебным. Также здесь реализована логика для загрузки переменных окружения и работы с морфологическим анализатором Natasha.

4. tbot.py

Основной модуль, который отвечает за запуск бота. Он инициализирует экземпляр бота, регистрирует обработчики из модуля handlers.py и запускает асинхронный цикл для обработки сообщений.

5. “.env”

Файл конфигурации, в котором хранятся чувствительные данные, такие как API-токены для Telegram и Hugging Face. Это позволяет избежать хранения таких данных в исходном коде и упрощает настройку бота для разных окружений.

**Преимущества модулярности:**

- Упрощение поддержки: Каждый модуль отвечает за свою часть функциональности, что упрощает внесение изменений и исправление ошибок.

- Повторное использование кода: Вспомогательные функции из `utils.py` могут быть легко использованы в других проектах.

- Тестируемость: Модули могут быть протестированы независимо друг от друга, что упрощает процесс тестирования.

- Масштабируемость: Добавление новой функциональности (например, новых команд или моделей) не требует изменения существующего кода, достаточно добавить новый модуль или расширить существующий.

## **2.10. Использование файла `educational\_keywords.txt`**

Для ограничения функциональности чат-бота только учебными темами был использован файл `educational\_keywords.txt`. Этот файл содержит список ключевых слов, связанных с учебным процессом, таких как "математика", "программирование", "лабораторная работа" и т.д. Когда пользователь отправляет сообщение, бот проверяет, содержит ли текст сообщения ключевые слова из этого списка. Если сообщение не содержит таких слов, бот отвечает, что может обрабатывать только учебные запросы.

**Как это работает:**

1. Загрузка ключевых слов:

При запуске бота ключевые слова загружаются из файла `educational\_keywords.txt` в память. Каждое слово приводится к нормализованной форме с использованием морфологического анализатора Natasha, чтобы учесть различные формы слов (например, "математика", "математике", "математику").

2. Проверка сообщений:

Когда пользователь отправляет сообщение, текст сообщения разбивается на слова, и каждое слово проверяется на соответствие ключевым словам из списка. Если хотя бы одно слово совпадает, сообщение считается учебным, и бот продолжает обработку запроса. В противном случае бот отвечает, что может обрабатывать только учебные вопросы.

**Преимущества:**

- Ограничение тематики: Бот фокусируется только на учебных вопросах, что помогает избежать обработки нерелевантных запросов.

- Гибкость: Список ключевых слов можно легко расширить или изменить, добавив новые слова в файл `educational\_keywords.txt`.

**Недостатки и костыльность решения:**

- Ограниченная точность: Это решение является костыльным, так как оно полагается на простой поиск ключевых слов. Оно не учитывает контекст сообщения, что может привести к ложным срабатываниям. Например, если пользователь напишет "Я не хочу учить математику", бот может посчитать это учебным запросом, хотя это не так.

- Зависимость от ключевых слов: Если пользователь использует синонимы или формулировки, которые не включены в список ключевых слов, бот может не распознать запрос как учебный.

- Отсутствие семантического анализа: Бот не понимает смысл сообщения, а только ищет совпадения по ключевым словам. Это ограничивает его способность корректно обрабатывать сложные запросы.

# **Структура проекта**



## **Модуль tbot.py**

1. Импорт библиотек:

- import asyncio: Импортирует библиотеку asyncio , которая позволяет работать с асинхронным программированием в Python.

- from telebot.async\_telebot import AsyncTeleBot : Импортирует класс AsyncTeleBot из библиотеки telebot, который используется для создания асинхронного Telegram-бота.

- from handlers import register\_handlers : Импортирует функцию register\_handlers , которая содержит обработчики команд и сообщений для бота.

- from utils import load\_env\_vars : Импортирует функцию load\_env\_vars , которая используется для загрузки переменных окружения.

2. Загрузка переменных окружения:

API\_TOKEN = load\_env\_vars("TELEGRAM\_BOT\_TOKEN")

Здесь вызывается функция load\_env\_vars, передавая ей имя переменной окружения ("TELEGRAM\_BOT\_TOKEN"), которая должна содержать токен API для доступа к Telegram Bot API. Токен сохраняется в переменной API\_TOKEN.

3. Инициализация бота:

bot = AsyncTeleBot(API\_TOKEN)

Создается экземпляр бота bot, используя загруженный токен API. Теперь бот может взаимодействовать с Telegram API.

4. Регистрация обработчиков:

register\_handlers(bot)

Вызывает функцию register\_handlers, передавая ей экземпляр бота. Эта функция отвечает за регистрацию всех обработчиков команд и событий, которые будут обрабатывать входящие сообщения от пользователей.

5. Определение основной асинхронной функции:

async def main() -> None:

"""Запуск бота."""

await bot.polling()

Здесь определяется асинхронная функция main, которая отвечает за запуск бота. Метод await bot.polling() запускает процесс опроса Telegram API, чтобы получать обновления и обрабатывать их в соответствии с зарегистрированными обработчиками.

6. Запуск программы:

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

Эта конструкция проверяет, выполняется ли скрипт напрямую. Если это так, запускается асинхронная функция main с помощью asyncio.run(). Это позволяет начать выполнение бота.

## **Модуль handlers.py**

1. Импорт необходимых модулей:

- from typing import Dict, List : Импортирует типы Dict и List для аннотации типов, что позволяет более точно описывать структуры данных.

- from telebot import types : Импортирует типы из библиотеки telebot, которые могут использоваться для работы с сообщениями и кнопками в Telegram.

- from telebot.async\_telebot import AsyncTeleBot : Импортирует асинхронный класс бота для работы с Telegram API.

- from models import set\_model, generate\_response, current\_model : Импортирует функции для установки модели, генерации ответов и получения текущей модели.

- from utils import create\_menu\_keyboard, is\_educational\_message : Импортирует вспомогательные функции для создания клавиатуры меню и проверки, является ли сообщение образовательным.

2. Пример расписания:

schedule: Dict[str, List[str]] = {

"Понедельник": ["Математика - 10:00", "Физика - 12:00"],

"Вторник": ["Программирование - 9:00", "Английский - 14:00"],

}

Здесь создается словарь schedule, который сопоставляет дни недели (ключи) со списками занятий (значения) для каждого дня.

3. Пример дедлайнов:

deadlines: Dict[str, str] = {

"Лабораторная работа по Python": "2023-12-15",

"Экзамен по математике": "2023-12-20",

}

Этот словарь deadlines хранит названия дедлайнов (ключи) и их соответствующие даты (значения).

4. Словарь для сопоставления названий кнопок с именами моделей:

model\_mapping: Dict[str, str] = {

"LLaMA": "meta-llama/Llama-3.2-3B-Instruct",

"ruGPT-3": "ai-forever/rugpt3small\_based\_on\_gpt2",

}

Этот словарь model\_mapping сопоставляет названия моделей (ключи) с их идентификаторами или ссылками (значения).

def register\_handlers(bot: AsyncTeleBot) -> None:

- Функция называется register\_handlers и принимает один аргумент bot, который является экземпляром асинхронного бота AsyncTeleBot.

- Аннотация -> None указывает, что функция не возвращает никакого значения.

3. Обработчик команды /start:

@bot.message\_handler(commands=['start'])

async def send\_welcome(message: types.Message) -> None:

- Декоратор @bot.message\_handler(commands=['start']) указывает, что следующая функция будет вызываться при получении команды /start.

- Функция send\_welcome принимает объект сообщения message, который содержит информацию о команде, отправленной пользователем.

Тело функции:

await bot.send\_message(

message.chat.id,

"Привет! Я твой помощник для учебы. Чем могу помочь?",

reply\_markup=create\_menu\_keyboard(),

)

- Эта функция отправляет приветственное сообщение пользователю, используя метод send\_message.

- message.chat.id используется для указания получателя сообщения.

- Сообщение включает текст приветствия и вызывает функцию create\_menu\_keyboard(), которая создает и возвращает клавиатуру меню для взаимодействия с пользователем.

Обработчик текстового сообщения "Расписание":

@bot.message\_handler(func=lambda message: message.text == "Расписание")

async def show\_schedule(message: types.Message) -> None:

- Декоратор @bot.message\_handler(func=lambda message: message.text == "Расписание") указывает, что следующая функция будет вызываться, когда пользователь отправляет сообщение с текстом "Расписание".

- Функция show\_schedule также принимает объект сообщения message.

Тело функции:

today = "Понедельник" # Замените на реальное определение дня

if today in schedule:

await bot.send\_message(

message.chat.id,

f"Расписание на сегодня:\n" + "\n".join(schedule[today]),

)

else:

await bot.send\_message(message.chat.id, "Сегодня занятий нет.")

- Здесь определяется переменная today, которая в данном случае жестко задана как "Понедельник". В реальной реализации это значение может определяться динамически (например, с использованием библиотеки для работы с датами).

- Если текущий день находится в расписании (словаре schedule), бот отправляет сообщение с расписанием на сегодня. Для этого используется "\n".join(schedule[today]), чтобы объединить все занятия в один текст с переносами строк.

- Если занятий нет, бот отправляет сообщение, информирующее пользователя об этом.

Обработчик для команды "Напоминания"

@bot.message\_handler(func=lambda message: message.text == "Напоминания")

async def show\_deadlines(message: types.Message) -> None:

reminders = [f"{task}: {date}" for task, date in deadlines.items()]

await bot.send\_message(

message.chat.id,

"Ваши дедлайны:\n" + "\n".join(reminders),

)

- Декоратор: @bot.message\_handler(func=lambda message: message.text == "Напоминания") указывает, что этот обработчик будет вызван, когда пользователь отправляет сообщение с текстом "Напоминания".

- Асинхронная функция: async def show\_deadlines(message: types.Message) -> None определяет асинхронную функцию, которая принимает объект сообщения message от пользователя.

- Формирование списка напоминаний:

reminders = [f"{task}: {date}" for task, date in deadlines.items()]

Здесь используется списковое включение для создания списка строк, в которых каждому заданию (task) сопоставляется его дата (date) из словаря deadlines. Таким образом, создается список строк в формате "Задание: Дата".

- Отправка сообщения:

await bot.send\_message(

message.chat.id,

"Ваши дедлайны:\n" + "\n".join(reminders),

)

Бот отправляет сообщение пользователю с заголовком "Ваши дедлайны:" и списком всех напоминаний, соединённых переносами строки.

Обработчик для команды "Выбрать модель"

@bot.message\_handler(func=lambda message: message.text == "Выбрать модель")

async def choose\_model(message: types.Message) -> None:

markup = types.ReplyKeyboardMarkup(row\_width=2)

markup.add(\*[types.KeyboardButton(model\_name) for model\_name in model\_mapping.keys()])

await bot.send\_message(

message.chat.id,

"Выберите модель для ответов на вопросы:",

reply\_markup=markup,

)

- Декоратор: @bot.message\_handler(func=lambda message: message.text == "Выбрать модель") указывает, что этот обработчик будет вызван, когда пользователь отправляет сообщение с текстом "Выбрать модель".

- Асинхронная функция: async def choose\_model(message: types.Message) -> None определяет асинхронную функцию, принимающую объект сообщения message.

- Создание клавиатуры:

markup = types.ReplyKeyboardMarkup(row\_width=2)

markup.add(\*[types.KeyboardButton(model\_name) for model\_name in model\_mapping.keys()])

- Создается объект markup, представляющий клавиатуру с кнопками. Параметр row\_width=2 устанавливает количество кнопок в строке.

- Затем создаются кнопки для каждой модели из словаря model\_mapping, и они добавляются на клавиатуру.

- Отправка сообщения с клавиатурой:

await bot.send\_message(

message.chat.id,

"Выберите модель для ответов на вопросы:",

reply\_markup=markup,

)

Бот отправляет сообщение пользователю с просьбой выбрать модель для ответов и прикрепляет созданную клавиатуру.

Обработчик выбора модели

@bot.message\_handler(func=lambda message: message.text in model\_mapping.keys())

async def handle\_model\_selection(message: types.Message) -> None:

- Декоратор: @bot.message\_handler(func=lambda message: message.text in model\_mapping.keys()) означает, что этот обработчик будет вызван, когда пользователь отправит сообщение, текст которого совпадает с одним из ключей в словаре model\_mapping.

- Асинхронная функция: async def handle\_model\_selection(message: types.Message) -> None определяет асинхронную функцию, которая принимает объект сообщения message.

selected\_model\_name = model\_mapping.get(message.text, None)

- Здесь используется метод get для извлечения имени модели из словаря model\_mapping по тексту сообщения пользователя. Если модель не найдена, возвращается None.

if selected\_model\_name:

set\_model(selected\_model\_name)

await bot.send\_message(

message.chat.id,

f"Теперь используется модель {message.text}.",

reply\_markup=create\_menu\_keyboard(),

)

- Если модель была найдена (т.е. selected\_model\_name не равен None), вызывается функция set\_model(selected\_model\_name), чтобы установить выбранную модель. Затем бот отправляет сообщение пользователю, информируя его о том, что выбранная модель теперь используется, и прикрепляет клавиатуру меню.

else:

await bot.send\_message(

message.chat.id,

"Неизвестная модель. Пожалуйста, выберите модель из списка.",

reply\_markup=create\_menu\_keyboard(),

)

- Если модель не найдена, бот отправляет сообщение с просьбой выбрать модель из списка, также прикрепляя меню.

Обработчик текстовых сообщений

@bot.message\_handler(func=lambda message: True)

async def handle\_message(message: types.Message) -> None:

- Декоратор: @bot.message\_handler(func=lambda message: True) означает, что этот обработчик будет вызван для всех текстовых сообщений, так как условие всегда истинно (возвращает True).

- Асинхронная функция: async def handle\_message(message: types.Message) -> None определяет асинхронную функцию, принимающую объект сообщения message.

user\_input = message.text

- Получает текст сообщения от пользователя и сохраняет его в переменной user\_input.

if is\_educational\_message(user\_input):

- Вызывается функция is\_educational\_message, чтобы проверить, связано ли сообщение с учебой. Если это так, выполняется следующее:

try:

await bot.send\_chat\_action(message.chat.id, "typing")

- Здесь бот отправляет статус "typing" (печатает), информируя пользователя о том, что идет обработка его запроса.

response = generate\_response(user\_input)

await bot.send\_message(message.chat.id, f"Ответ:\n{response}")

- Вызывается функция generate\_response(user\_input) для генерации ответа на основе пользовательского ввода. Затем бот отправляет сгенерированный ответ пользователю.

except Exception as e:

await bot.send\_message(message.chat.id, f"Ошибка при запросе к модели: {str(e)}")

- Если во время обработки сообщения возникает ошибка, бот отправляет сообщение с информацией об ошибке.

else:

await bot.send\_message(message.chat.id, "Пожалуйста, задавайте вопросы, связанные с учебой.")

- Если сообщение не связано с учебой, бот отправляет сообщение с просьбой задавать только соответствующие вопросы.

## **Модуль models.py**

from huggingface\_hub import InferenceClient

- Импорт InferenceClient из библиотеки huggingface\_hub для взаимодействия с моделями через API.

from utils import load\_env\_vars

- Импорт функции load\_env\_vars, которая загружает переменные окружения.

from transformers import pipeline

- Импорт функции pipeline из библиотеки transformers, упрощающей использование моделей машинного обучения.

HUGGING\_FACE\_API\_TOKEN = load\_env\_vars("HUGGING\_FACE\_API\_TOKEN")

- Загружает API-токен для Hugging Face, необходимый для аутентификации.

client = InferenceClient(api\_key=HUGGING\_FACE\_API\_TOKEN)

- Создает клиент для взаимодействия с моделями через API, используя загруженный токен.

current\_model: str = "meta-llama/Llama-3.2-3B-Instruct"

- Устанавливает начальную модель, которая будет использоваться для генерации текста.

rugpt\_pipe = pipeline(

"text-generation",

model="ai-forever/rugpt3small\_based\_on\_gpt2",

truncation=True, # Добавлено для обрезки текста

do\_sample=True # Добавлено для использования temperature

)

- Инициализирует pipeline для модели ruGPT3small, с возможностью обрезки текста и использования температуры для генерации.

def set\_model(model\_name: str) -> None:

global current\_model

current\_model = model\_name

- Функция для настройки текущей модели, изменяющая значение current\_model.

def generate\_response(prompt: str) -> str:

if current\_model == "ai-forever/rugpt3small\_based\_on\_gpt2":

response = rugpt\_pipe(prompt, max\_length=700, num\_return\_sequences=1, temperature=0.7)

return response[0]['generated\_text']

else:

response = client.text\_generation(

prompt=prompt,

model=current\_model,

max\_new\_tokens=700,

temperature=0.7,

top\_k=50,

do\_sample=True, # Убедитесь, что do\_sample=True

)

return response

- Функция generate\_response генерирует текст на основе запроса пользователя (prompt).

- Если текущая модель — ruGPT3small, используется локальный pipeline.

- В противном случае используется InferenceClient для генерации текста через API, с указанными параметрами, такими как temperature и top\_k.

## **Модуль utils.py**

1. Импорт библиотек:

- os: модуль для взаимодействия с операционной системой, используется для работы с переменными окружения.

- Optional, List из typing: используются для аннотации типов, что помогает сделать код более понятным.

- load\_dotenv из dotenv: функция для загрузки переменных окружения из файла .env.

- ReplyKeyboardMarkup, KeyboardButton из telebot.types: используются для создания клавиатур в Telegram-ботах.

- MorphVocab из natasha: библиотека для морфологического анализа русского языка.

- unicodedata: модуль для работы с данными юникода.

- logging: модуль для ведения логов, используется для отладки.

2. Настройка логирования:

- logging.basicConfig(level=logging.DEBUG) : устанавливает уровень логирования на DEBUG, что позволяет выводить сообщения отладки.

3. Инициализация морфологического словаря:

- morph\_vocab = MorphVocab() : создается экземпляр морфологического словаря, который может быть использован для анализа и обработки русского языка.

4. Функция load\_env\_vars:

- Эта функция загружает переменную окружения по заданному ключу.

- Внутри функции сначала вызывается load\_dotenv(".env") для загрузки переменных окружения из файла .env. Затем используется os.getenv(key) для получения значения переменной по ключу.

Функция create\_menu\_keyboard():

- Эта функция создает клавиатуру для пользователя с основными командами.

- Возвращает объект типа ReplyKeyboardMarkup, который представляет собой клавиатуру с кнопками.

- markup = ReplyKeyboardMarkup(row\_width=2) : создается новый объект ReplyKeyboardMarkup, который представляет клавиатуру. Параметр row\_width=2 указывает, что кнопки будут располагаться по два в каждой строке.

- markup.add(...) : метод add используется для добавления кнопок на клавиатуру. Внутри этого метода создаются кнопки с текстом:

- "Расписание"

- "Напоминания"

- "Поиск материалов"

- "Задать вопрос"

- "Выбрать модель"

- return markup : в конце функция возвращает созданный объект клавиатуры, который можно использовать в сообщении для пользователя.

Функция normalize\_text(text: str) -> str

- Эта функция нормализует текст, чтобы обеспечить согласованное сравнение строк.

- text: строка, которую необходимо нормализовать.

- Возвращает нормализованную строку.

- unicodedata.normalize('NFKC', text) : используется метод normalize из модуля unicodedata, который нормализует текст по стандарту Unicode. Формат `'NFKC'` (Normalization Form KC) означает, что текст будет преобразован в каноническую форму, где различные графемы, представляющие один и тот же символ, будут приведены к одной форме. Это важно для сравнения строк, так как оно устраняет различия в представлении одного и того же символа.

Чтение ключевых слов из файла:

try:

with open("educational\_keywords.txt", "r", encoding="utf-8") as file:

educational\_keywords = [

normalize\_text(line.strip().lower()) for line in file.readlines() if line.strip()

]

except FileNotFoundError:

logging.error("Файл educational\_keywords.txt не найден.")

return False

- Используется блок try-except для обработки возможного исключения, если файл educational\_keywords.txt не найден.

- Открывается файл в режиме чтения с кодировкой UTF-8.

- Считываются все строки из файла, каждая строка очищается от пробелов (с помощью strip()) и приводится к нижнему регистру (с помощью lower()), затем нормализуется с помощью функции normalize\_text.

- Если файл не найден, в лог записывается ошибка, и функция возвращает False.

Проверка на пустой список ключевых слов:

if not educational\_keywords:

logging.error("Файл с ключевыми словами пуст.")

return False

- Если файл успешно открыт, но не содержит ключевых слов, также записывается ошибка в лог, и функция возвращает False.

Нормализация текста сообщения:

message\_text = normalize\_text(message\_text.lower())

- Текст сообщения приводится к нижнему регистру и нормализуется.

Разбиение сообщения на слова:

user\_words = message\_text.split()

- Сообщение разбивается на отдельные слова, которые сохраняются в списке user\_words.

Проверка каждого слова:

for word in user\_words:

parsed\_word = morph\_vocab(word) # Вернуть список объектов MorphForm

- Для каждого слова выполняется морфологический анализ с использованием объекта morph\_vocab, который возвращает список объектов MorphForm.

Проверка нормальных форм:

if parsed\_word:

normal\_forms = [form.normal\_form for form in parsed\_word if hasattr(form, 'normal\_form')]

- Если parsed\_word не пуст, вытаскиваются нормальные формы слов (леммы) и сохраняются в списке normal\_forms.

Логирование и проверка:

for normal\_form in normal\_forms:

normal\_form = normalize\_text(normal\_form)

logging.debug(f"Word: {word}, Normal Form: {normal\_form}")

if normal\_form in educational\_keywords:

return True

- Каждая нормальная форма нормализуется и записывается в лог для отладки.

- Если нормальная форма слова содержится в списке ключевых слов, функция возвращает True.

Возврат значения по умолчанию:

return False

- Если ни одно из слов не совпало с ключевыми словами, функция возвращает False.

# **Работа бота**

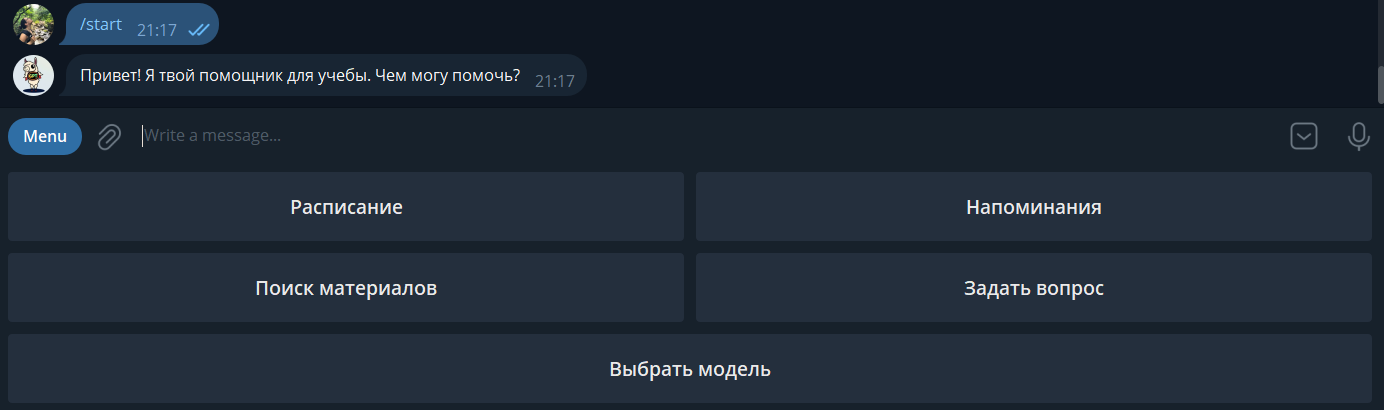


Рисунок . Стартовое сообщение бота

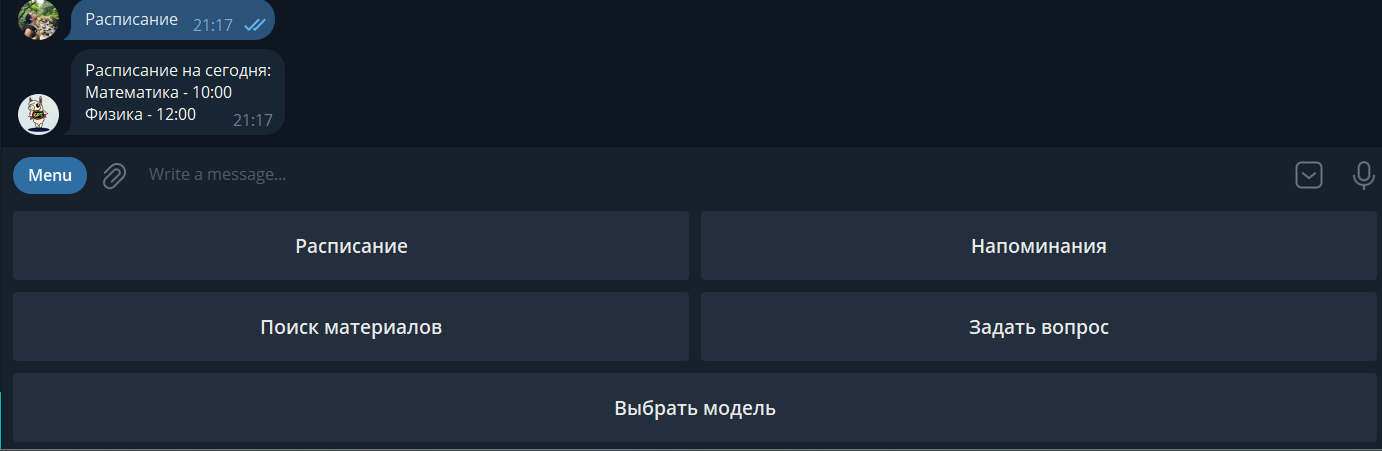


Рисунок . Показ расписания

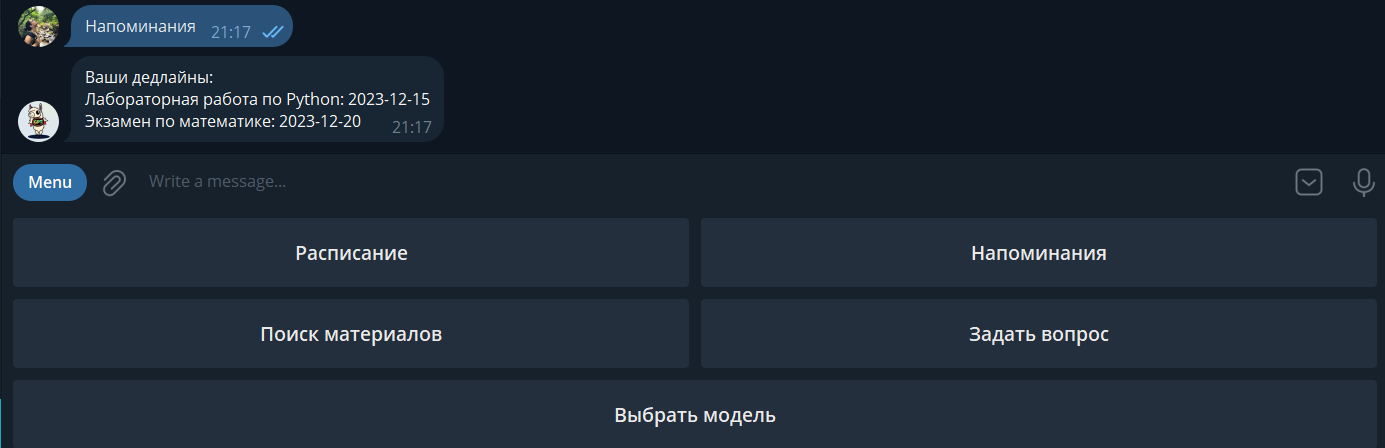


Рисунок . Напоминания о дедлайнах

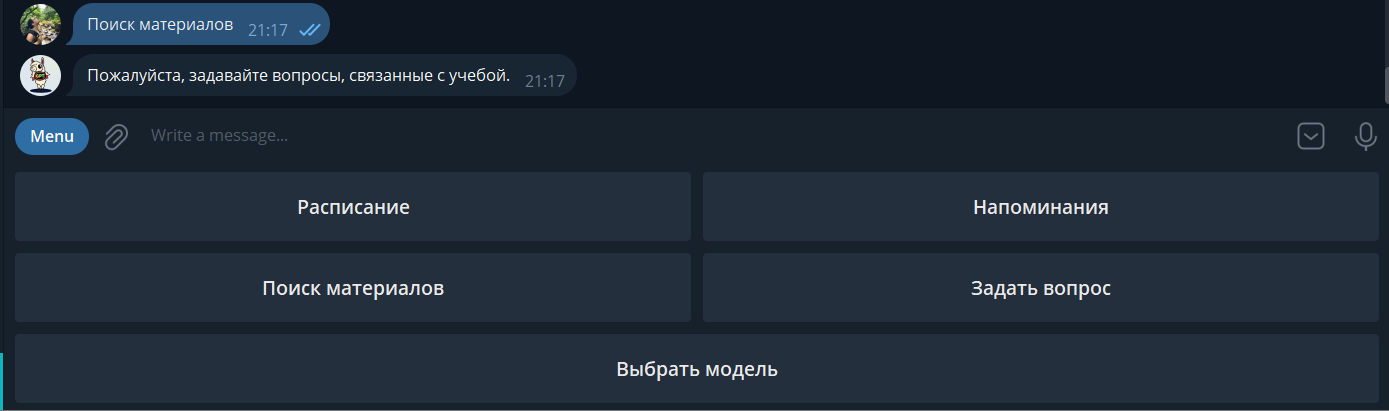


Рисунок . Поиск материалов переводит на вопрос к модели

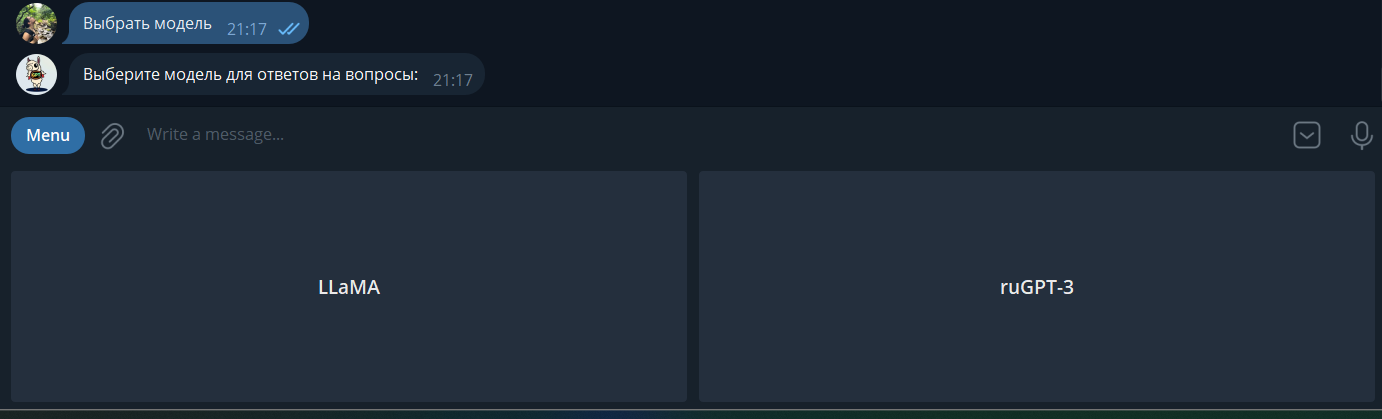


Рисунок . Экран выбора модели

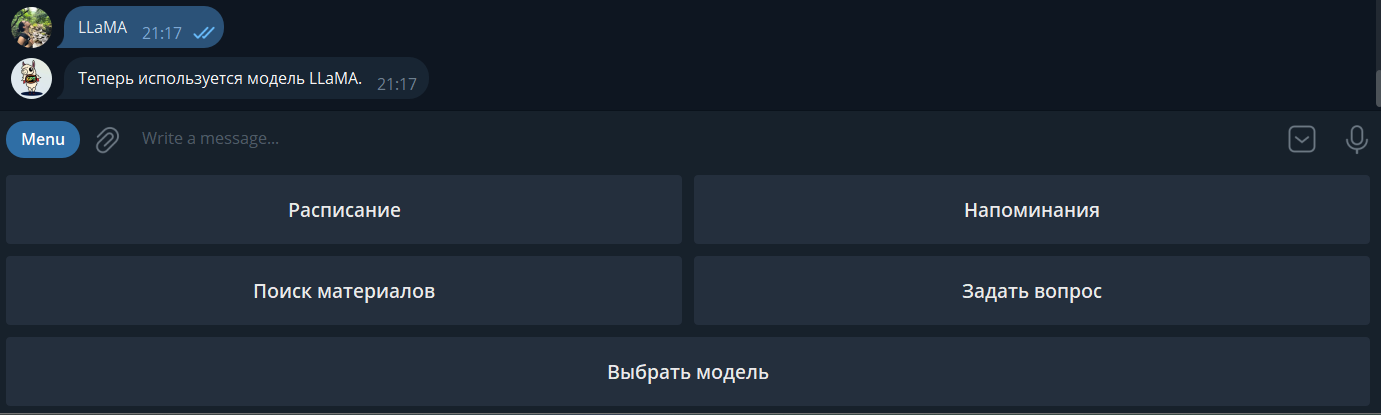


Рисунок . Выбор модели Llama-3.2-3B-Instruct

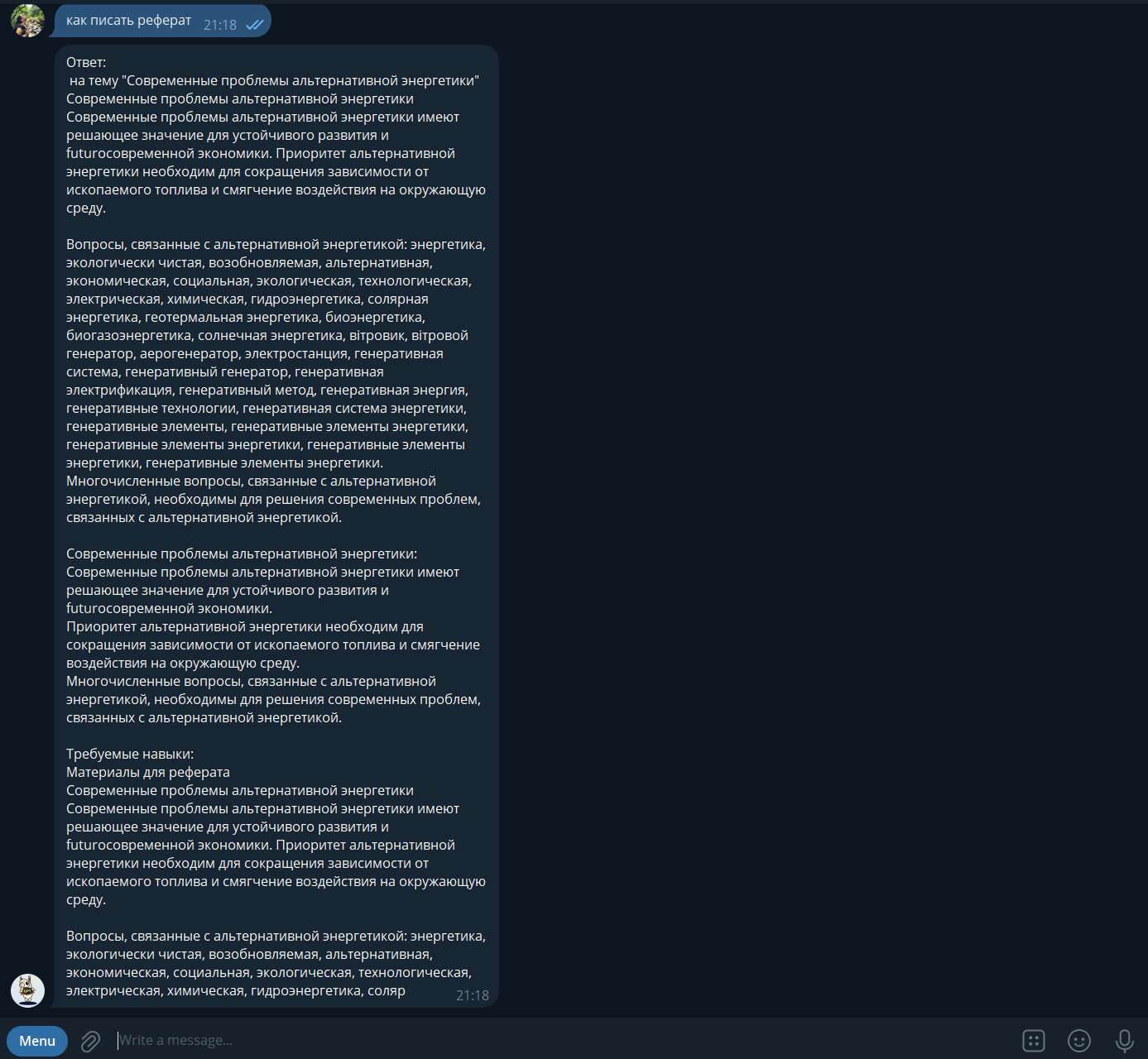


Рисунок . Ответ модели Llama-3.2-3B-Instruct

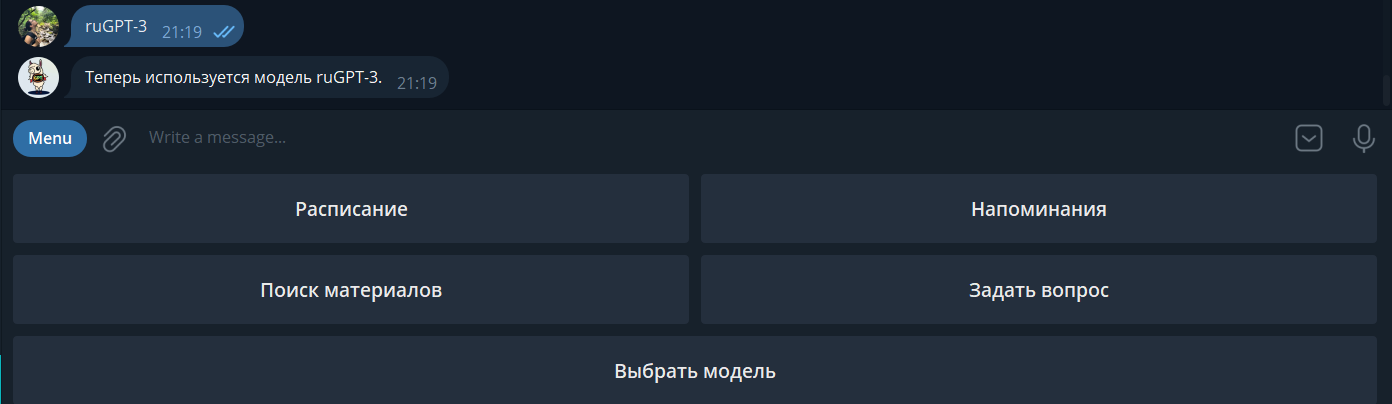


Рисунок 8. Выбор модели ruGPT3small\_based\_on\_gpt2

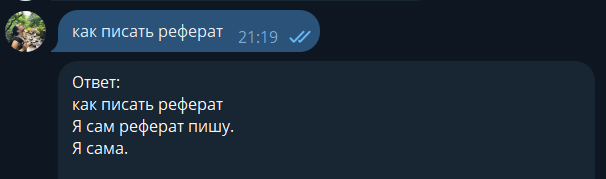


Рисунок 9. Ответ модели ruGPT3small\_based\_on\_gpt2

# 

# **ВЫВОДЫ**

Чат-бот успешно справляется с поставленными задачами и может быть использован в учебном процессе для помощи студентам. Асинхронный подход позволяет эффективно обрабатывать запросы пользователей, а использование языковых моделей GPT и LLaMA обеспечивает высокое качество генерации ответов. Однако, для улучшения качества ответов на русском языке требуется тонкая настройка моделей, что требует значительных вычислительных ресурсов.

При сравнении ответов моделей rugpt3small\_based\_on\_gpt2 и Llama-3.2-3B-Instruct были выявлены большие недостатки модели rugpt3small\_based\_on\_gpt2, связанные со странными и неструктурированными ответами. Было протестировано много моделей gpt, однако в данном случае только модель Llama-3.2-3B-Instruct показала себя хорошо при ответе на учебные вопросы.

# **ИСПОЛЬЗУЕМАЯ ЛИТЕРАТУРА**

1. AsyncTeleBot Documentation – [https://github.com/eternnoir/pyTelegramBotAPI]

2. Hugging Face Transformers – [https://huggingface.co/docs/transformers/index]

3. Natasha: Russian NLP Library – [https://github.com/natasha/natasha]

4. Python Asyncio Documentation – [https://docs.python.org/3/library/asyncio.html]

5. PEP 8 — Style Guide for Python Code – [https://www.python.org/dev/peps/pep-0008/]

6. Hugging Face Model Hub – [https://huggingface.co/models]

7. https://school.kontur.ru/publications/2567

# **ПРИЛОЖЕНИЕ А. Код: модуль tbot.py**

import asyncio

from telebot.async\_telebot import AsyncTeleBot

from handlers import register\_handlers

from utils import load\_env\_vars

# Загрузка переменных окружения

API\_TOKEN = load\_env\_vars("TELEGRAM\_BOT\_TOKEN")

# Инициализация бота

bot = AsyncTeleBot(API\_TOKEN)

# Регистрация обработчиков

register\_handlers(bot)

async def main() -> None:

"""Запуск бота."""

await bot.polling()

if \_\_name\_\_ == "\_\_main\_\_":

asyncio.run(main())

# **ПРИЛОЖЕНИЕ Б. Код: модуль models.py**

from huggingface\_hub import InferenceClient

from utils import load\_env\_vars

from transformers import pipeline

# Загрузка переменных окружения

HUGGING\_FACE\_API\_TOKEN = load\_env\_vars("HUGGING\_FACE\_API\_TOKEN")

# Инициализация Hugging Face Inference Client

client = InferenceClient(api\_key=HUGGING\_FACE\_API\_TOKEN)

# Переменная для хранения выбранной модели

current\_model: str = "meta-llama/Llama-3.2-3B-Instruct" # Исходная модель

# Инициализация ruGPT3small для использования

rugpt\_pipe = pipeline(

"text-generation",

model="ai-forever/rugpt3small\_based\_on\_gpt2",

truncation=True, # Добавлено для обрезки текста

do\_sample=True # Добавлено для использования temperature

)

def set\_model(model\_name: str) -> None:

"""

Устанавливает текущую модель для генерации ответов.

Аргументы:

model\_name (str): Название модели.

"""

global current\_model

current\_model = model\_name

def generate\_response(prompt: str) -> str:

"""

Генерирует ответ на запрос пользователя с использованием текущей модели.

Аргументы:

prompt (str): Запрос пользователя.

Возвращает:

str: Ответ модели.

"""

if current\_model == "ai-forever/rugpt3small\_based\_on\_gpt2":

response = rugpt\_pipe(prompt, max\_length=700, num\_return\_sequences=1, temperature=0.7)

return response[0]['generated\_text']

else:

response = client.text\_generation(

prompt=prompt,

model=current\_model,

max\_new\_tokens=700,

temperature=0.7,

top\_k=50,

do\_sample=True, # Убедитесь, что do\_sample=True

)

return response

# **ПРИЛОЖЕНИЕ В. Код: модуль handlers.py**

from typing import Dict, List

from telebot import types

from telebot.async\_telebot import AsyncTeleBot

from models import set\_model, generate\_response, current\_model

from utils import create\_menu\_keyboard, is\_educational\_message

# Пример расписания

schedule: Dict[str, List[str]] = {

"Понедельник": ["Математика - 10:00", "Физика - 12:00"],

"Вторник": ["Программирование - 9:00", "Английский - 14:00"],

}

# Пример дедлайнов

deadlines: Dict[str, str] = {

"Лабораторная работа по Python": "2023-12-15",

"Экзамен по математике": "2023-12-20",

}

# Словарь для сопоставления названий кнопок с именами моделей

model\_mapping: Dict[str, str] = {

"LLaMA": "meta-llama/Llama-3.2-3B-Instruct",

"ruGPT-3": "ai-forever/rugpt3small\_based\_on\_gpt2", # Обновлено на новую модель

}

def register\_handlers(bot: AsyncTeleBot) -> None:

"""

Регистрирует обработчики команд и сообщений.

Аргументы:

bot (AsyncTeleBot): Экземпляр бота.

"""

@bot.message\_handler(commands=['start'])

async def send\_welcome(message: types.Message) -> None:

# Обработчик команды /start

await bot.send\_message(

message.chat.id,

"Привет! Я твой помощник для учебы. Чем могу помочь?",

reply\_markup=create\_menu\_keyboard(),

)

@bot.message\_handler(func=lambda message: message.text == "Расписание")

async def show\_schedule(message: types.Message) -> None:

# Обработчик нажатия на "Расписание"

today = "Понедельник" # Замените на реальное определение дня

if today in schedule:

await bot.send\_message(

message.chat.id,

f"Расписание на сегодня:\n" + "\n".join(schedule[today]),

)

else:

await bot.send\_message(message.chat.id, "Сегодня занятий нет.")

@bot.message\_handler(func=lambda message: message.text == "Напоминания")

async def show\_deadlines(message: types.Message) -> None:

# Обработчик нажатия на "Напоминания"

reminders = [f"{task}: {date}" for task, date in deadlines.items()]

await bot.send\_message(

message.chat.id,

"Ваши дедлайны:\n" + "\n".join(reminders),

)

@bot.message\_handler(func=lambda message: message.text == "Выбрать модель")

async def choose\_model(message: types.Message) -> None:

# Обработчик нажатия на "Выбрать модель"

markup = types.ReplyKeyboardMarkup(row\_width=2)

markup.add(\*[types.KeyboardButton(model\_name) for model\_name in model\_mapping.keys()])

await bot.send\_message(

message.chat.id,

"Выберите модель для ответов на вопросы:",

reply\_markup=markup,

)

@bot.message\_handler(func=lambda message: message.text in model\_mapping.keys())

async def handle\_model\_selection(message: types.Message) -> None:

# Обработчик выбора модели

selected\_model\_name = model\_mapping.get(message.text, None)

if selected\_model\_name:

set\_model(selected\_model\_name)

await bot.send\_message(

message.chat.id,

f"Теперь используется модель {message.text}.",

reply\_markup=create\_menu\_keyboard(),

)

else:

await bot.send\_message(

message.chat.id,

"Неизвестная модель. Пожалуйста, выберите модель из списка.",

reply\_markup=create\_menu\_keyboard(),

)

@bot.message\_handler(func=lambda message: True)

async def handle\_message(message: types.Message) -> None:

# Обработчик текстовых сообщений

user\_input = message.text

if is\_educational\_message(user\_input):

try:

await bot.send\_chat\_action(message.chat.id, "typing")

response = generate\_response(user\_input)

await bot.send\_message(message.chat.id, f"Ответ:\n{response}")

except Exception as e:

await bot.send\_message(message.chat.id, f"Ошибка при запросе к модели: {str(e)}")

else:

await bot.send\_message(message.chat.id, "Пожалуйста, задавайте вопросы, связанные с учебой.")

# **ПРИЛОЖЕНИЕ Г. Код: модуль utils.py**

import os

from typing import Optional, List

from dotenv import load\_dotenv

from telebot.types import ReplyKeyboardMarkup, KeyboardButton

from natasha import MorphVocab

import unicodedata

import logging

logging.basicConfig(level=logging.DEBUG)

# Инициализация морфологического словаря

morph\_vocab = MorphVocab()

def load\_env\_vars(key: str) -> str:

"""

Загружает переменную окружения по ключу.

Аргументы:

key (str): Ключ переменной окружения.

Возвращает:

str: Значение переменной окружения.

Исключения:

ValueError: Если переменная окружения не найдена.

"""

load\_dotenv(".env")

value = os.getenv(key)

if value is None:

raise ValueError(f"Переменная окружения {key} не найдена.")

return value

def create\_menu\_keyboard() -> ReplyKeyboardMarkup:

"""

Создает клавиатуру с основными командами для пользователя.

Возвращает:

ReplyKeyboardMarkup: Объект клавиатуры с кнопками.

"""

markup = ReplyKeyboardMarkup(row\_width=2)

markup.add(

KeyboardButton("Расписание"),

KeyboardButton("Напоминания"),

KeyboardButton("Поиск материалов"),

KeyboardButton("Задать вопрос"),

KeyboardButton("Выбрать модель"),

)

return markup

def normalize\_text(text: str) -> str:

"""

Нормализовать текст, чтобы обеспечить согласованное сравнение.

"""

return unicodedata.normalize('NFKC', text)

def is\_educational\_message(message\_text: str) -> bool:

"""

Проверяет, связано ли сообщение с учебой, используя список ключевых слов и Natasha для учета склонений.

"""

# Чтение ключевых слов из файла

try:

with open("educational\_keywords.txt", "r", encoding="utf-8") as file:

educational\_keywords = [

normalize\_text(line.strip().lower()) for line in file.readlines() if line.strip()

]

except FileNotFoundError:

logging.error("Файл educational\_keywords.txt не найден.")

return False

if not educational\_keywords:

logging.error("Файл с ключевыми словами пуст.")

return False

# Приведение текста сообщения к нижнему регистру и нормализация

message\_text = normalize\_text(message\_text.lower())

# Разбиваем сообщение на слова

user\_words = message\_text.split()

for word in user\_words:

# Приводим слово к начальной форме (лемматизация)

parsed\_word = morph\_vocab(word) # Вернуть список объектов MorphForm

if parsed\_word: # Убедиться что список не пуст

# Вытаскиваем нормальные формы слова как string

normal\_forms = [form.normal\_form for form in parsed\_word if hasattr(form, 'normal\_form')]

# Проверить все нормальные формы слова

for normal\_form in normal\_forms:

normal\_form = normalize\_text(normal\_form)

# Логирование normal\_form для дебага

logging.debug(f"Word: {word}, Normal Form: {normal\_form}")

# Проверяем, есть ли начальная форма в списке ключевых слов

if normal\_form in educational\_keywords:

return True

return False