Н И Т У « М И С И С »

Кафедра инженерной кибернетики

ОТЧЕТ

ПО

ЛАБОРАТОРНОЙ РАБОТЕ №2

«Разработка асинхронного чат-бота для мессенджера Telegram с использованием языковых моделей архитектур GPT и LLaMA»

учебная дисциплина «Экспертные и рекомендательные системы»

Студент: Гаврилюк А.В. гр. БПМ-22-ПО-3

Преподаватель: Хонер П. Д.

Оценка: _____

Дата защиты: ____20.10.2025____

2025 г.

Введение

Цель данной лабораторной работы заключалась в том, чтобы разработать асинхронный Telegram чат-бот для решения прикладной задачи с использованием языковых моделей архитектур GPT и LLaMA. В процессе работы требовалось сравнить функциональные возможности этих сервисов, оценить качество генерируемых ответов каждым API в контексте специализированной задачи и сравнить полученные результаты между собой.

1. Поставленная задача

Суть задачи заключалась в разработке кулинарного чат-бота на базе мессенджера Telegram, который способен давать советы, рекомендации и рецепты пользователям, интересующимся кулинарией. Для реализации функционала генерации ответов необходимо было реализовать чат бота с использованием двух различных API больших языковых моделей (LLM) и сравнить их работу.

Основные требования к приложению:

- Возможность выбора между двумя различными языковыми моделями
- Специализация на кулинарной тематике
- Генерация ответов на русском языке
- Корректная обработка запросов пользователей

В качестве результата работы приложение предоставляет пользователю текстовые ответы от выбранной языковой модели, адаптированные под кулинарную тематику.

2. Выбранные средства разработки программного обеспечения

Для разработки приложения были использованы два API, доступ к которым был получен посредством использования платформы RapidAPI:

- **GPT-4 API**: данное API предоставляет доступ к языковой модели GPT-4, разработанной компанией OpenAI. Модель способна генерировать качественные текстовые ответы, понимать сложные запросы и поддерживать естественный диалог. Модель особенно хороша в понимании нюансов языка и генерации структурированных ответов. Недостатком этого API является невозможность вручную указать параметры языковой модели.
- **LLaMA3 API**: это API предоставляет доступ к языковой модели LLaMA3, разработанной компанией Meta. Модель является open-source решением и показывает хорошие результаты в генерации текстов. Недостатком этого API является невозможность вручную указать параметры языковой модели.

В качестве языка разработки был использован Python. Для создания Telegram-бота использовалась асинхронная библиотека aiogram. Для выполнения асинхронных HTTP-запросов к API была использована библиотека aiohttp.

3. Описание и примеры исходных данных

Современные языковые модели имеют множество настраиваемых параметров, которые влияют на качество и характер генерируемых ответов:

Temperature - параметр, контролирующий случайность и креативность ответов. Значения варьируются от 0 до 2:

- Низкие значения (0.0 - 0.3) делают ответы более предсказуемыми и сфокусированными.
- Средние значения (0.5 - 0.8) обеспечивают баланс между креативностью и точностью.
- Высокие значения (1.0 - 2.0) увеличивают разнообразие и креативность, но могут снижать релевантность.

Top-p (Nucleus Sampling) - параметр, определяющий из какого процента наиболее вероятных токенов модель будет выбирать следующее слово. Значения от 0 до 1, стандартное значение 0.9 - 0.95.

Max Tokens - максимальная длина генерируемого ответа в токенах. Один токен примерно равен одному слову или части слова.

Frequency Penalty - штраф за частое повторение одних и тех же слов или фраз. Значения от -2.0 до 2.0.

Presence Penalty - штраф за повторение уже упомянутых в диалоге тем. Помогает модели исследовать новые направления беседы.

System Message - начальная инструкция, задающая роль и контекст работы модели. В данном проекте используется для специализации на кулинарной тематике.

Web Access - параметр, определяющий возможность модели обращаться к актуальной информации из интернета.

В рамках данной работы основное внимание уделялось системному сообщению (контексту) для специализации бота на кулинарной тематике, а остальные параметры использовались со значениями по умолчанию, установленными провайдером API, так как возможности передать их вручную провайдером API предусмотрено не было.

4. Описание и примеры работы приложения

Приложение представляет собой Telegram-бота, который при запуске предоставляет пользователю информацию о своих возможностях и примеры использования. Пользователь может:

1. Задавать вопросы о кулинарии, просить рецепты и советы.
2. Переключаться между двумя языковыми моделями с помощью команды /setmodel
3. Получать ответы на русском языке без излишнего форматирования.

Примеры типичных запросов к боту:

- Посоветуй рецепт шоколадного печенья.
- Я люблю итальянскую кухню, но пицца и паста надоели — что попробовать еще?
- Как правильно варить пасту?
- Какие специи подходят к курице?

Бот корректно обрабатывает длинные ответы, автоматически разбивая их на части, если они превышают ограничения Telegram по длине сообщения (4000 символов).

При попытке пользователя задать вопросы, не относящиеся к кулинарии, бот деликатно направляет диалог обратно к кулинарной тематике благодаря специально настроенному системному контексту.

Также, на рисунке 1 продемонстрирована процедура смены модели, а также асинхронность работы чат-бота. На «Привет» API дало ответ быстрее, хотя данный запрос был отправлен вторым по счету.

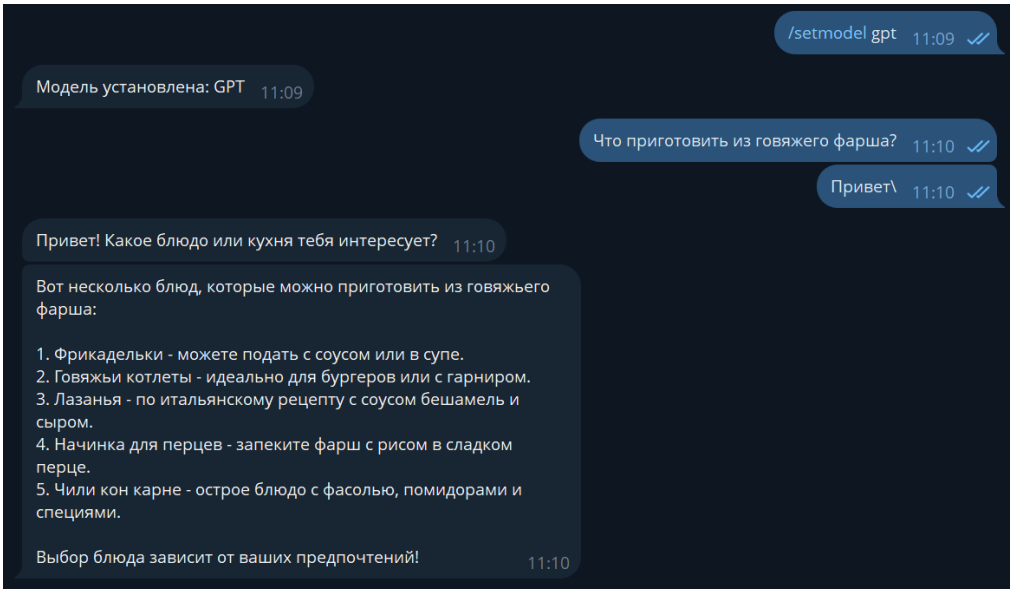

Рис. 1. Демонстрация работы чат бота.

5. Результаты

В ходе тестирования оба API показали довольно хорошие результаты в решении поставленной задачи. Ниже представлены основные наблюдения по работе двух API:

GPT-4 API:

- Генерирует более структурированные и детальные ответы.
- Лучше понимает сложные многосоставные запросы.
- Более точно следует инструкциям из системного контекста.
- Ответы отличаются высокой релевантностью.

LLaMA3 API:

- Показывает хорошее качество генерации текстов, но хуже GPT-4 API.
- Справляется с базовыми кулинарными запросами.
- Обеспечивает приемлемую скорость ответа.
- Является альтернативным решением с открытым исходным кодом.

Оба API успешно справились с задачей создания кулинарного ассистента. GPT-4 показал несколько лучшие результаты в понимании контекста и качестве ответов, однако LLaMA3 также продемонстрировал хорошую производительность и может служить достойной альтернативой.

6. Исходный код

bot.py

```
import asyncio
from aiogram import Bot, Dispatcher
from aiogram.types import Message
from aiogram.filters import Command, CommandStart
from api_handler import get_gpt4_response, get_llama3_response
from config import get_config

config = get_config()
bot = Bot(token=config["tg_bot_token"])
dp = Dispatcher()

# Модель по умолчанию
current_model = "gpt4"
```

```python
# Задаем контекст
CULINARY_CONTEXT = (
    "You are a professional culinary assistant and gastronomic advisor. "
    "You help users find interesting dishes and cuisines based on their tastes. "
    "Avoid unnecessary details — respond concisely and only about cooking and food. "
    "If user asks general or irrelevant question, gently guide them back to culinary
topics."
    "Only use russian language to answer, do not use complicated text formatting, do
not use highlighting."
)


@dp.message(CommandStart())
async def start_command(message: Message):
    await message.answer(
        "Привет! Я твой кулинарный советчик.\n\n"
        "Могу быть полезным во всех кулинарных темах, могу что-то посоветовать или
даже дать фирменный рецепт!\n\n"
        "Популярные запросы:\n"
        "Посоветуй рецепт шоколадного печенья.\n"
        "«Я люблю итальянскую кухню, но пицца и паста надоели — что попробовать
еще?»\n\n"
        "Также, можешь выбрать модель, с помощью которой я буду думать:\n"
        "/setmodel gpt — GPT\n"
        "/setmodel llama — LLAMA"
    )


@dp.message(Command("setmodel"))
async def set_model(message: Message):
    global current_model
    parts = message.text.split()
    if len(parts) != 2:
        return await message.answer("Выбрать модель можно так: /setmodel gpt или
/setmodel llama")
    model = parts[1].lower().strip()
    if model in ["gpt", "llama"]:
        current_model = model
        await message.answer(f"Модель установлена: {current_model.upper()}")
    else:
        await message.answer("Неизвестная модель. Попробуй команды /setmodel gpt или
/setmodel llama.")


@dp.message()
async def handle_user_query(message: Message):
    print(message.text)
    query = message.text.strip()
    try:
        if current_model == "llama":
            answer = await get_llama3_response(query, CULINARY_CONTEXT)
        else:
            answer = await get_gpt4_response(query, CULINARY_CONTEXT)

        if not answer:
            answer = "Не удалось получить ответ от API."

        # Длинные ответы разбиваем
        max_len = 4000
        for i in range(0, len(answer), max_len):
            await message.answer(answer[i:i + max_len])

    except Exception as e:
        await message.answer(f"Ошибка при обработке запроса: {e}")
```

```
async def main():
    await dp.start_polling(bot)

if __name__ == "__main__":
    asyncio.run(main())
```

api_handler.py

```
from typing import Any

import aiohttp
from config import get_config

config = get_config()


async def query_llm_api(url: str, payload: dict) -> Any | None:
    """
    Асинхронный POST-запрос к RapidAPI LLM.
    """
    headers = {
        "x-rapidapi-key": config["rapidapi_key"],
        "x-rapidapi-host": config["rapidapi_host"],
        "Content-Type": "application/json"
    }

    try:
        async with aiohttp.ClientSession() as session:
            async with session.post(url, headers=headers, json=payload,timeout=ai-
ohttp.ClientTimeout(total=120)) as resp:
                resp.raise_for_status()
                return await resp.json()
    except Exception as e:
        print(f"[API ERROR] {e}")
        return None


async def get_gpt4_response(user_query: str, context: str = "") -> str:
    """
    GPT-4 API через RapidAPI
    """
    payload = {
        "messages": [
            {"role": "system", "content": context},
            {"role": "user", "content": user_query}
        ],
        "web_access": False
    }

    response = await query_llm_api(config["gpt4_url"], payload)
    if response and "result" in response:
        return response["result"]
    return "Не удалось получить ответ от GPT-4 API."


async def get_llama3_response(user_query: str, context: str = "") -> str:
    """
    LLaMA3 API через RapidAPI
    """
    payload = {
        "messages": [
            {"role": "system", "content": context},
            {"role": "user", "content": user_query}
        ],
        "web_access": False
```

```
    }

    response = await query_llm_api(config["llama3_url"], payload)
    print(response["result"])
    if response and "result" in response:
        return response["result"]
    return "Не удалось получить ответ от LLaMA3 API."
```