МИНИСТЕРСТВО ТРАНСПОРТА РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

«РОССИЙСКИЙ УНИВЕРСИТЕТ ТРАНСПОРТА»

(РУТ (МИИТ))

Кафедра «Цифровые технологии управления транспортными процессами»

Институт управления и цифровых технологий

КУРСОВАЯ РАБОТА

по дисциплине: «Системы искусственного интеллекта»
на тему: Разработка приложения для взаимодействия с большими языковыми моделями

Выполнила:
студент 4 курса
группы УВПв-421
Гатауллина Л.И.
Преподаватель:
Варнавский А.Н.

Москва

2025

Содержание

ВВЕДЕНИЕ 3
ОСНОВНАЯ ЧАСТЬ 4
1. Выбор инструментов и технологий 4
2. GigaChat 4
3. Локальная модель 5
4. Возможности приложения 6
5. Графический интерфейс 6
6. Работа приложения 8
7. Сравнение моделей 9
ЗАКЛЮЧЕНИЕ 12
ПРИЛОЖЕНИЕ А 13

ВВЕДЕНИЕ

Развитие технологий искусственного интеллекта в последние годы привело к широкому внедрению языковых моделей (Large Language Models, LLM) в различные сферы жизни.

Целью данной курсовой работы является разработка графического приложения, взаимодействующего с двумя типами языковых моделей: облачной моделью GigaChat и локальной моделью Ollama, а также обращение к двум моделям и сравнение вариантов ответа.

ОСНОВНАЯ ЧАСТЬ

1. Выбор инструментов и технологий

Программное приложение с графическим интерфейсом было написано на Python с использованием библиотеки tkinter для интерфейса, код в приложении А.

Для обработки и анализа данных используется Pandas, для построения графиков Matplotlib, хранение истории запросов в JSON формате. Формат JSON был выбран из-за его простоты, читаемости и поддержки в Python через стандартную библиотеку. Так как работаем с русским языком, важно отметить, что при сохранении данных в JSON используется параметр ensure_ascii=False. Это гарантирует, что данные будут записаны в читаемом виде, без экранирования символов

Для обеспечения отзывчивости интерфейса используется многопоточность. При отправке вопроса основной поток остается свободным, а обработки запроса выполняются в отдельном потоке. Используется threading.Thread для асинхронного выполнения запросов к моделям.

Для повышения отказоустойчивости используются конструкции try-except, перехватывающие исключения. Это позволяет корректно информировать пользователя о проблемах в работе с моделью и сохранять стабильность интерфейса.

2. GigaChat

Облачная LLM от Сбера – GigaChat. Для использования GigaChat API, зарегистрировалась на портале https://developers.sber.ru, получила токен доступа, который вставлен в GIGACHAT_CREDENTIALS в коде. Для вызова облачной модели GIgaChat используем команду установки pip install gigachat на рисунке 1. Для вызова используется метод call_gigachat (question).

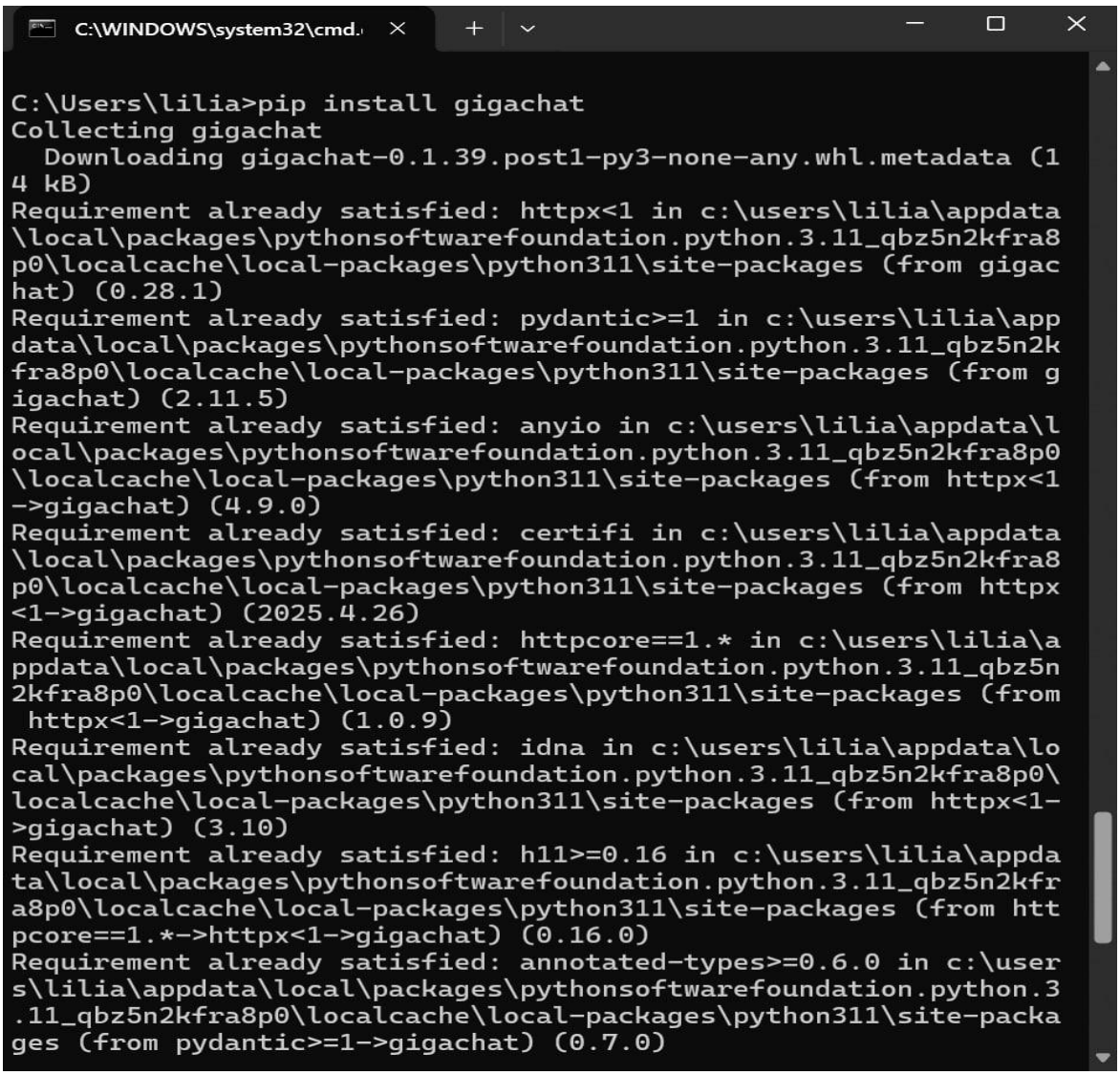

Рисунок 1 – Установка облачной модели GigaChat.

3. Локальная модель

Фреймворк для запуска локальных моделей Ollama.Для взаимодействия с локальной языковой моделью используем команду pip install ollama. Библиотека Ollama работает с одноименным сервером, который был установлен с официального сайта: https://ollama.ai. Для использования модели llama3 используем команду на рисунке 2. Для вызова локальной модели используется метод call_ollamat (question),

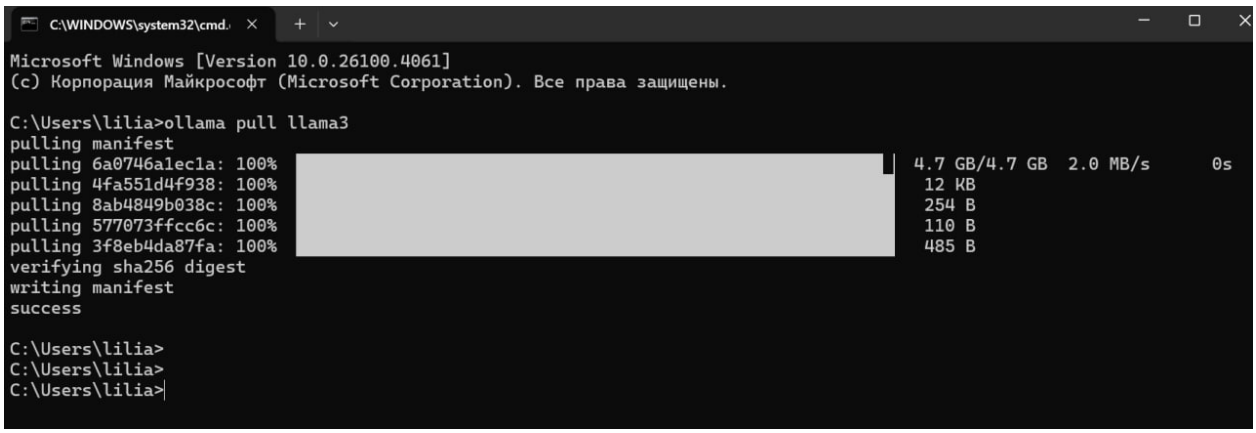

Рисунок 2 – Установка модели llama3.

4. Возможности приложения

Приложение представляет следующие возможности:

- Отправка вопроса пользователем.
- Получение ответа от GigaChat или локальной модели (Ollama).
- Возможность выбора режима:
  - Только GigaChat;
  - Только Ollama;
  - Сравнение обеих моделей.
- Отображение длины и времени ответа.
- Сохранение истории.
- Построение графика сравнения.
- Отображение статистики в виде таблицы.

5. Графический интерфейс

Графический интерфейс реализован с помощью tkinter, имеет удобный интерфейс:

- Текстовое поле для ввода запроса.
- Выпадающий список выбора режима работы (GigaChat / Ollama / сравнение).
- Кнопка отправки.
- Область вывода ответа.

– Таблица статистики. Для обновления таблицы статистики в интерфейсе используется метод update_stats_table().

На рисунках 3 и 4 представлен исходный интерфейс с выбором LLM.

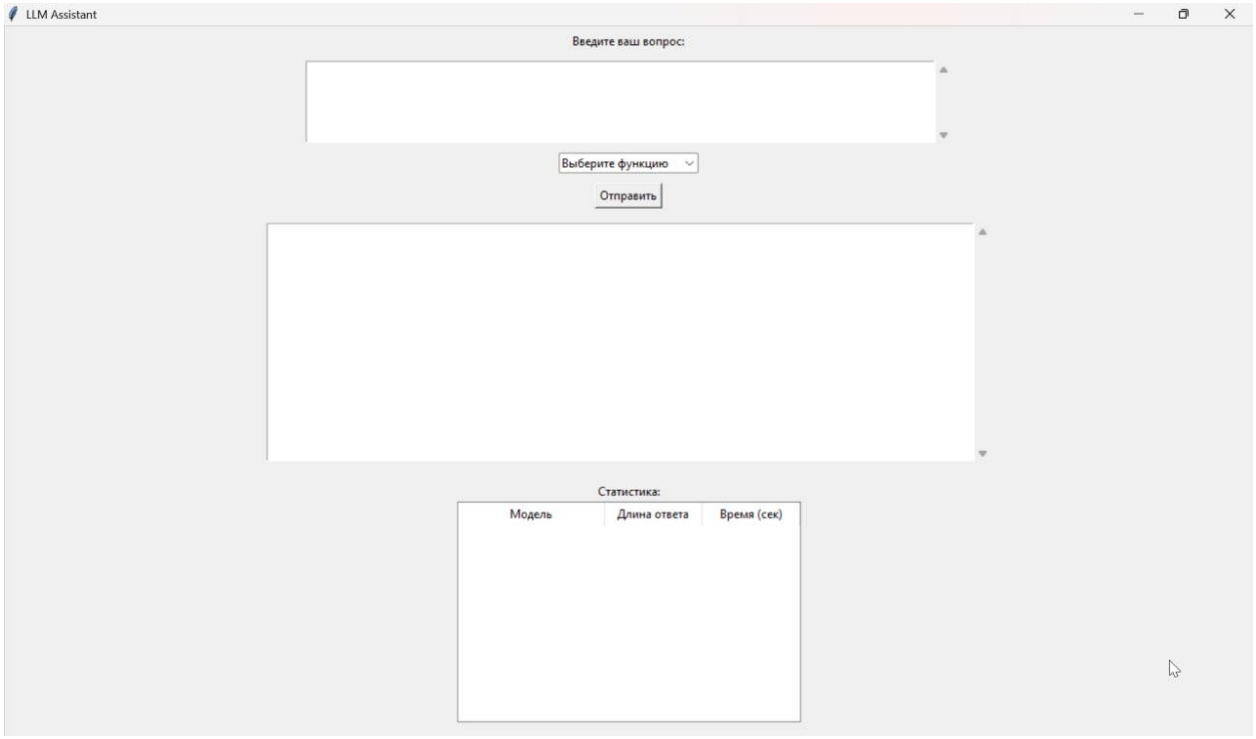

Рисунок 3 – Исходный интерфейс

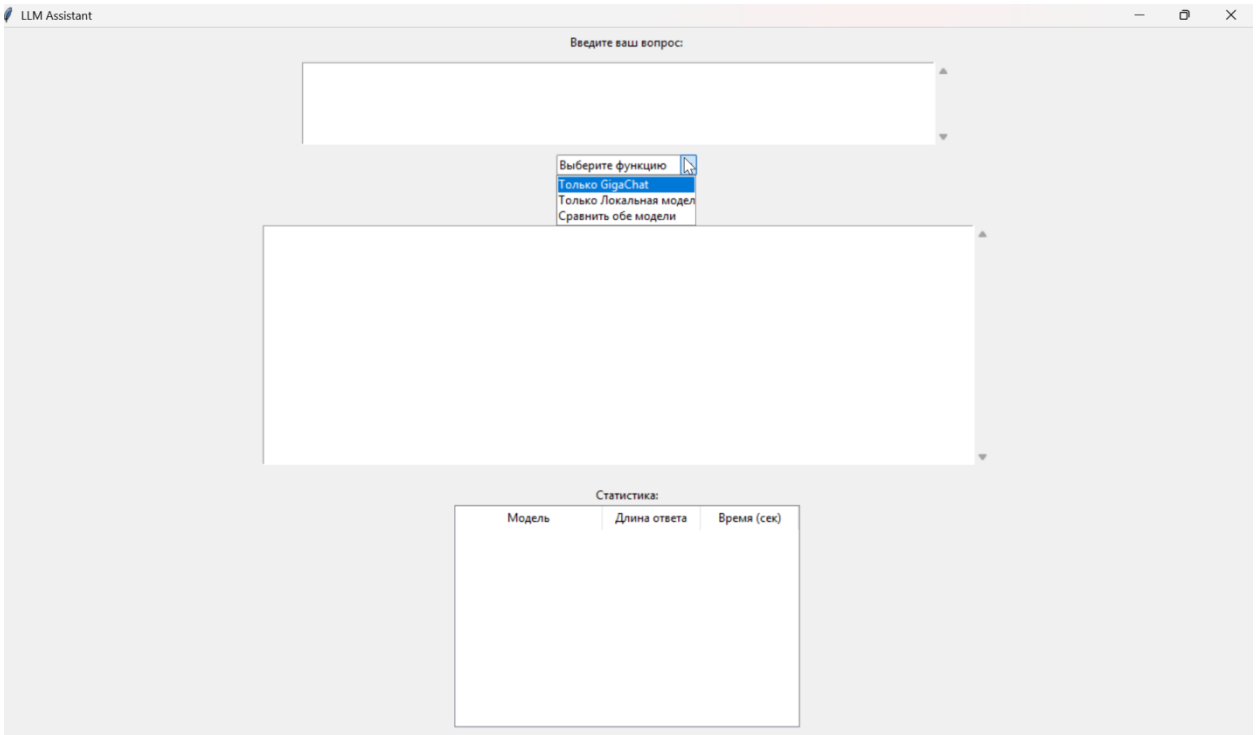

Рисунок 4 – Выбор языковой модели

6. Работа приложения

Режимы работы:

- В режиме «Только GigaChat» приложение получает ответ от сервера Сбера, вывод в таблице длины ответа и времени ответа.
- В режиме «Только Локальная модель» — модели Llama3, вывод в таблице длины ответа и времени ответа.
- В режиме «Сравнение» оба ответа выводятся подряд, сравнивается длина ответа и время ответа, строится гистограмма.

Для примера задаем вопрос: «Кто такой Александр Сергеевич Пушкин?». На рисунке 5 скрин запроса и вывода только GigaChat, длина ответа 1506 символов, время ответа примерно 3 сек. На рисунке 6 – вывод только локальной модели, длина ответа 1329 символов, примерно 134 сек. На рисунке 7 сравнение моделей, вывод обеих моделей.

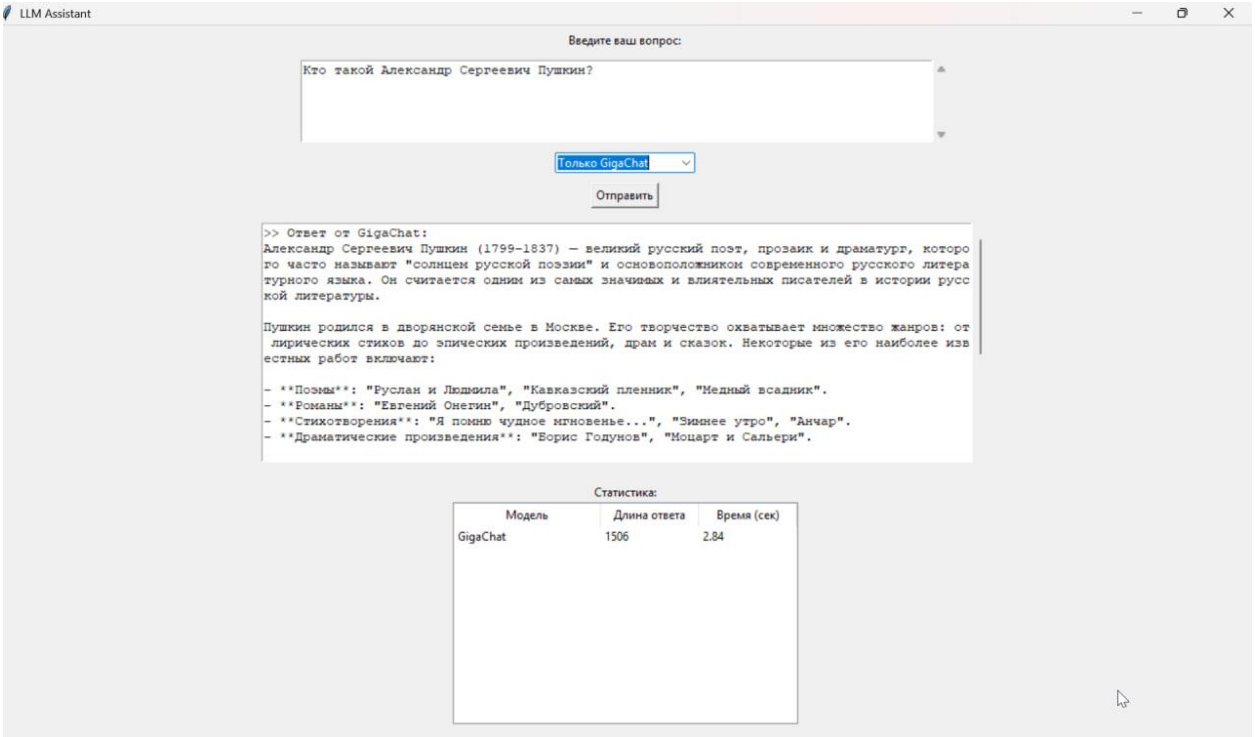

Рисунок 5 – Вывод GigaChat

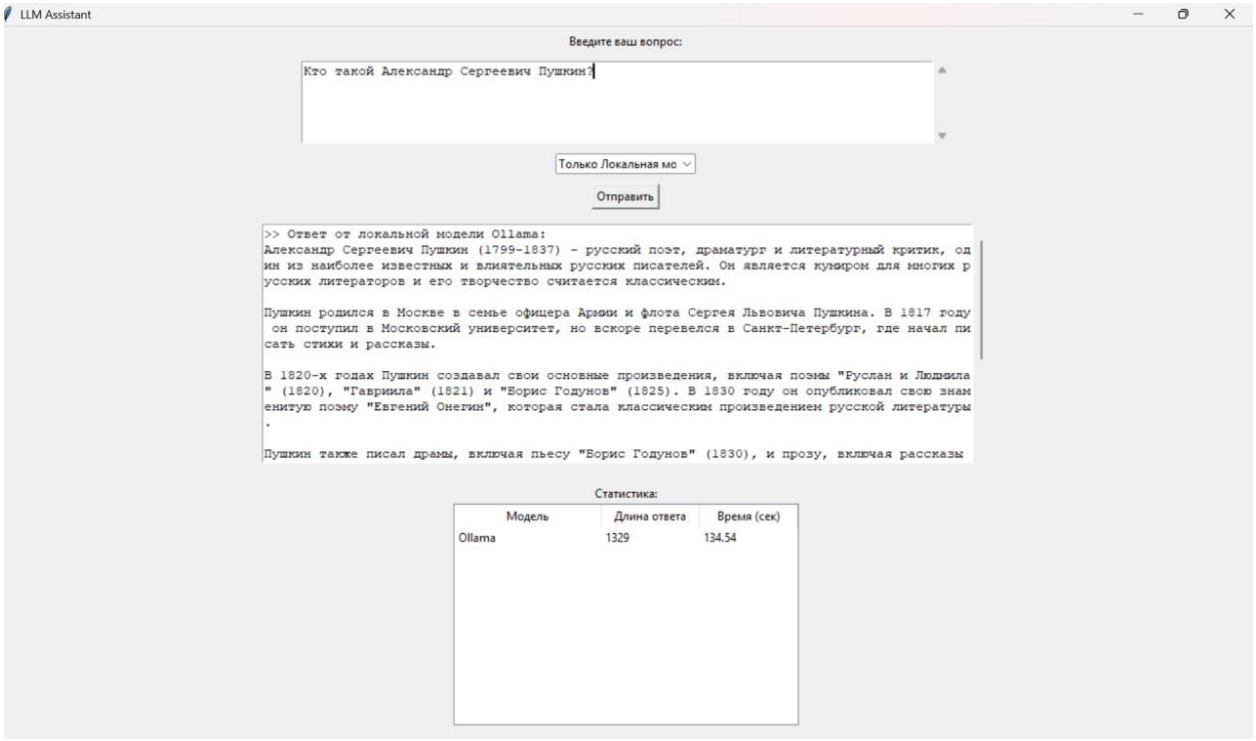

Рисунок 6 – Вывод локальной модели

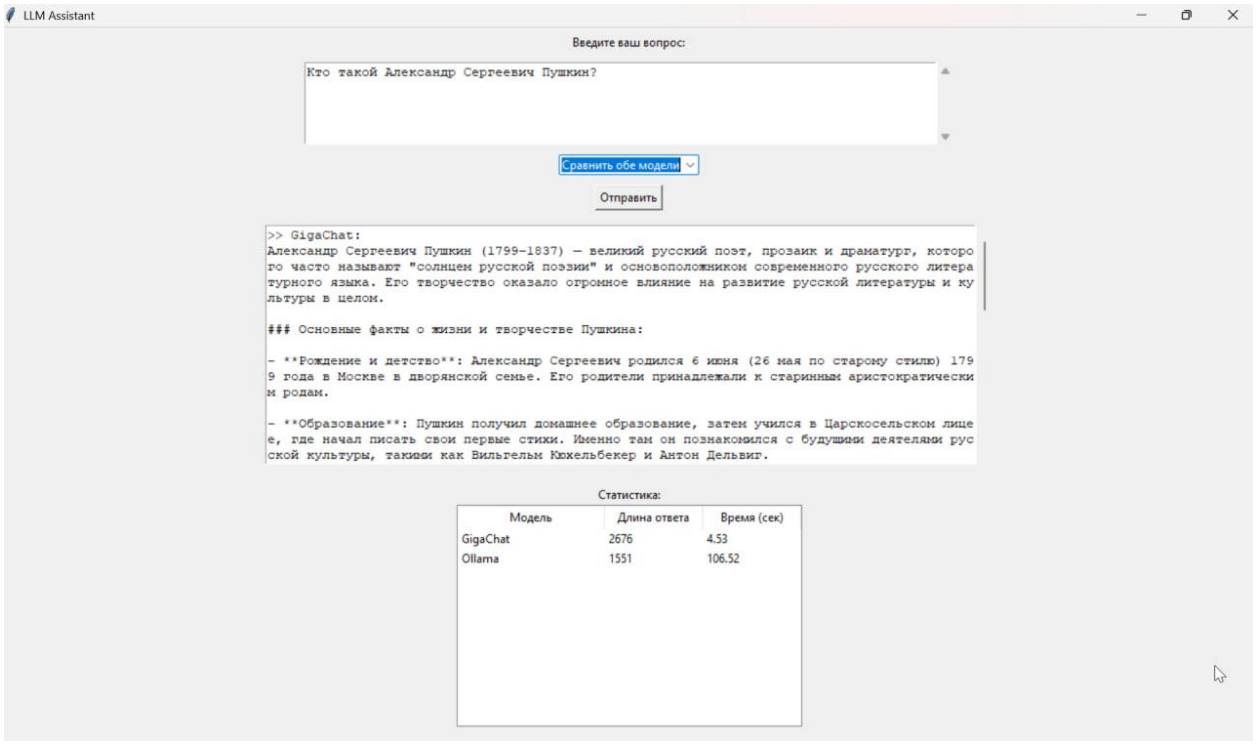

Рисунок 7 – Сравнение моделей

7. Сравнение моделей

Приложение реализует функцию сравнения двух моделей по длине ответа len(answer) и по времени выполнения time.time(). Метод сравнения в

коде compare_answers,при каждом сравнении добавляется новая запись в excel, обновляется таблица в интерфейсе, строится график сравнения.

Хранение данных:

– Вопросы и ответы сохраняются в history.json (рисунок 8). Метод export_history экспортирует историю в EXCEL.
– Сохранение числовых метрик: длина ответа и время выполнения в stats.xlsx (рисунок 9).

Для визуализации результатов используется matplotlib в функции plot_comparison. Длина и время ответов по каждой модели сохраняется в comparison.png (рисунок 10).

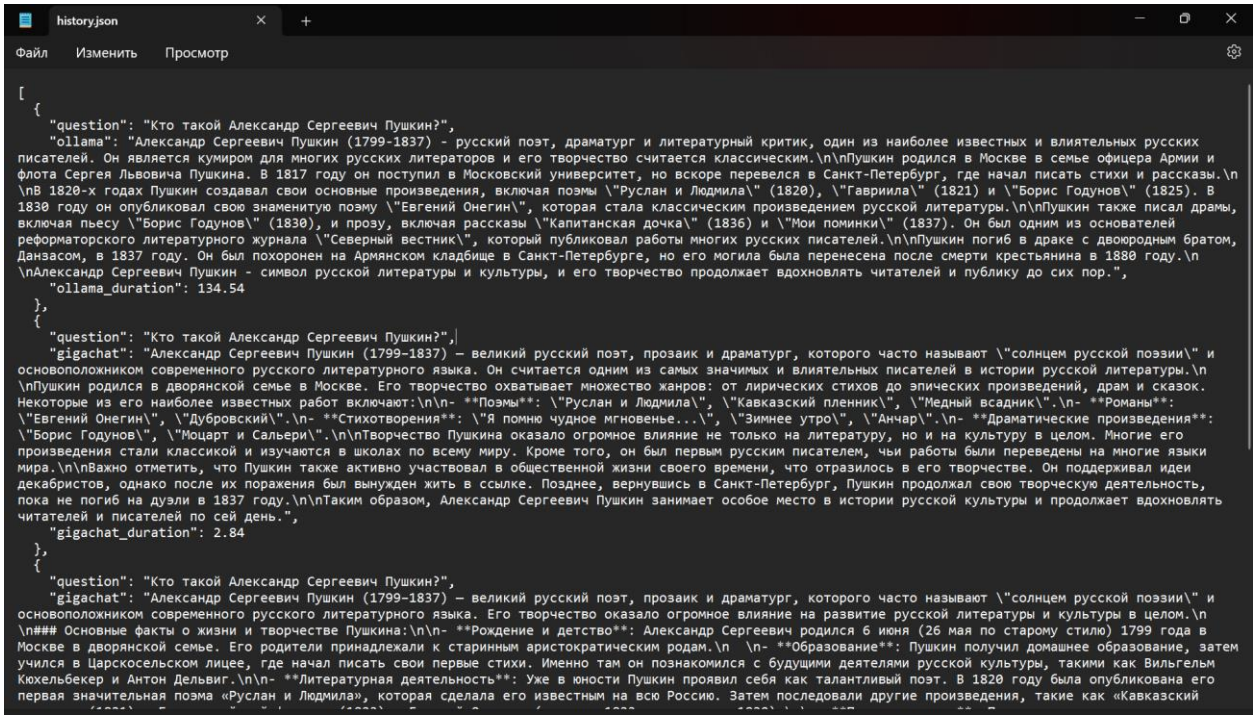

Рисунок 8 – history.json

	A	B	C	D	E
1	gigachat_len	ollama_len	gigachat_time	ollama_time	
2	2676	1551	4,5308442116	106,52373528	

Рисунок 9 – Данные моделей в EXCEL

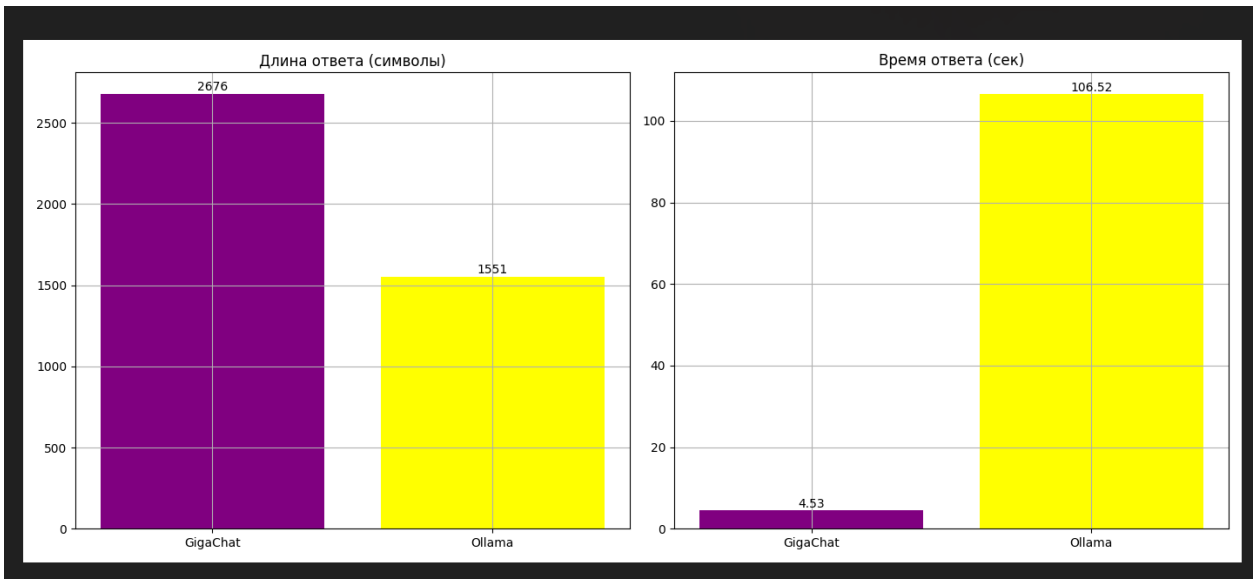

Рисунок 10 – График сравнения моделей

Из результатов сравнения видно, что GigaChat даёт более полный ответ и работает быстрее.

ЗАКЛЮЧЕНИЕ

В ходе выполнения курсовой работы было разработано приложение, взаимодействующее с двумя языковыми моделями: облачной (GigaChat) и локальной (Ollama). Была реализована отправка вопроса, получение ответа в графическом интерфейсе, автоматическое сравнение по длине и времени, сохранение истории, создание графиков и таблицы статистики в excel.

В результате сравнения был сделан вывод, что GigaChat даёт более подробные ответы, работает быстрее, в отличии от Ollama.

ПРИЛОЖЕНИЕ А

```
import tkinter as tk
from tkinter import ttk, scrolledtext, filedialog
import threading
import json
import os
import time
import matplotlib.pyplot as plt
import pandas as pd
import ollama
from gigachat import GigaChat, exceptions as gigachat_exceptions


# === Настройки ===
GIGACHAT_CREDENTIALS = {
    "credentials":
"NmQ0ODdlMWQtMzJiMC00OWI3LWJkMDMtYTBiZTIwMzdkZjVlOmQxZD
I2YjE2LTY2YTQtNDVjMC05MDMzLTUyZWM0MjA5YzYyOQ==",
    "scope": "GIGACHAT_API_PERS"
}
OLLAMA_MODEL = "llama3"

# === Файлы для хранения данных ===
history_file = "history.json"
stats_file = "stats.xlsx"

if not os.path.exists(history_file):
    with open(history_file, "w") as f:
        json.dump([], f)
```

```
# === Вспомогательные функции с замером времени ===
def call_gigachat(question):
    start_time = time.time()
    try:
        with GigaChat(**GIGACHAT_CREDENTIALS, verify_ssl_certs=False) as
giga:
            response = giga.chat(question)
            answer = response.choices[0].message.content
    except gigachat_exceptions.GigaChatException as e:
        raise Exception(f"Ошибка GigaChat: {e}")
    duration = time.time() - start_time
    return answer, duration



def call_ollama(question):
    start_time = time.time()
    try:
        response = ollama.chat(model=OLLAMA_MODEL, messages=[
            {'role': 'user', 'content': question},
        ])
        answer = response['message']['content']
    except Exception as e:
        raise Exception(f"Ошибка Ollama: {e}")
    duration = time.time() - start_time
    return answer, duration
```

```
# === Основное приложение ===
class LLMApp:
    def __init__(self, root):
        self.root = root
        self.root.title("LLM Assistant")
        self.root.geometry("1000x700")

        # === Переменные для статистики ===
        self.last_stats = {}

        self.history = []
        self.load_history()

        # === UI элементы ===
        self.question_label = tk.Label(root, text="Введите ваш вопрос:")
        self.question_label.pack(pady=5)

        self.question_entry = scrolledtext.ScrolledText(root, height=5)
        self.question_entry.pack(padx=10, pady=5)

        self.mode_var = tk.StringVar(value="Выберите функцию")
        self.mode_menu = ttk.Combobox(root, textvariable=self.mode_var,
                                      values=["Только GigaChat", "Только Локальная
модель", "Сравнить обе модели"])
        self.mode_menu.pack(pady=5)

        self.submit_button = tk.Button(root, text="Отправить",
command=self.start_query)
```

```
        self.submit_button.pack(pady=5)

        self.output_area = scrolledtext.ScrolledText(root, height=15, width=90)
        self.output_area.pack(padx=10, pady=10)

        # === Таблица статистики последнего запроса ===
        self.stats_frame = tk.Frame(root)
        self.stats_frame.pack(pady=10)

        self.stats_label = tk.Label(self.stats_frame, text="Статистика:")
        self.stats_label.pack()

        self.stats_table = ttk.Treeview(self.stats_frame, columns=("Модель",
"Длина", "Время"), show='headings')
        self.stats_table.heading("Модель", text="Модель")
        self.stats_table.heading("Длина", text="Длина ответа")
        self.stats_table.heading("Время", text="Время (сек)")
        self.stats_table.column("Модель", width=150)
        self.stats_table.column("Длина", width=100)
        self.stats_table.column("Время", width=100)
        self.stats_table.pack()

        self.export_button = tk.Button(root, text="Экспорт истории",
command=self.export_history)
        self.export_button.pack(pady=5)

    def load_history(self):
        with open(history_file, "r") as f:
            self.history = json.load(f)
```

```
    def save_history(self):
        with open(history_file, "w") as f:
            json.dump(self.history, f, ensure_ascii=False, indent=2)

    def start_query(self):
        self.submit_button.config(state=tk.DISABLED)
        self.output_area.delete(1.0, tk.END)
        self.output_area.insert(tk.END, "Обработка запроса...\n")
        threading.Thread(target=self.process_query).start()

    def process_query(self):
        question = self.question_entry.get("1.0", tk.END).strip()
        mode = self.mode_var.get()

        result = {}

        try:
            if mode == "Только GigaChat":
                answer, duration = call_gigachat(question)
                result = {"question": question, "gigachat": answer, "gigachat_duration":
round(duration, 2)}
                output = f">> Ответ от GigaChat:\n{answer}\n Время: {duration:.2f}
сек"
                self.update_last_stats("GigaChat", len(answer), duration)

            elif mode == "Только Локальная модель":
                answer, duration = call_ollama(question)
```

```
            result = {"question": question, "ollama": answer, "ollama_duration":
round(duration, 2)}
            output = f">> Ответ от локальной модели Ollama:\n{answer}\n
Время: {duration:.2f} сек"
            self.update_last_stats("Ollama", len(answer), duration)

        else:  # compare
            ans_gigachat, dur_gigachat = call_gigachat(question)
            ans_ollama, dur_ollama = call_ollama(question)

            result = {
                "question": question,
                "gigachat": ans_gigachat,
                "ollama": ans_ollama,
                "gigachat_duration": round(dur_gigachat, 2),
                "ollama_duration": round(dur_ollama, 2)
            }

            output = (
                f">> GigaChat:\n{ans_gigachat}\n Время: {dur_gigachat:.2f}
сек\n\n"
                f">> Локальная модель:\n{ans_ollama}\n Время: {dur_ollama:.2f}
сек"
            )

            self.compare_answers(ans_gigachat, ans_ollama, dur_gigachat,
dur_ollama)

        self.history.append(result)
        self.save_history()
```

```
            self.root.after(0, self.update_output, output)

        except Exception as e:
            self.root.after(0, self.update_output, f"❌Ошибка: {str(e)}")

    def update_output(self, text):
        self.output_area.delete(1.0, tk.END)
        self.output_area.insert(tk.END, text)
        self.submit_button.config(state=tk.NORMAL)

    def update_last_stats(self, model_name, length, duration):
        """Обновляет данные по последнему запросу"""
        self.last_stats = {model_name: (length, duration)}
        self.update_stats_table()

    def compare_answers(self, a1, a2, t1, t2):
        len1, len2 = len(a1), len(a2)
        time1, time2 = t1, t2

        # Сохраняем в Excel
        new_data = pd.DataFrame([{
            "gigachat_len": len1,
            "ollama_len": len2,
            "gigachat_time": time1,
            "ollama_time": time2
        }])

        if os.path.exists(stats_file):
```

```
            df_existing = pd.read_excel(stats_file)
            df_updated = pd.concat([df_existing, new_data], ignore_index=True)
        else:
            df_updated = new_data

        df_updated.to_excel(stats_file, index=False)

        # Обновляем статистику
        self.last_stats = {
            "GigaChat": (len1, time1),
            "Ollama": (len2, time2)
        }
        self.update_stats_table()
        self.plot_comparison(len1, len2, time1, time2)

    def update_stats_table(self):
        """Обновляет таблицу статистики"""
        self.stats_table.delete(*self.stats_table.get_children())

        for model, (length, duration) in self.last_stats.items():
            self.stats_table.insert("", "end", values=(model, length, f"{duration:.2f}"))

    def plot_comparison(self, len1, len2, time1, time2):
        labels = ['GigaChat', 'Ollama']
        lengths = [len1, len2]
        times = [time1, time2]

        fig, axes = plt.subplots(1, 2, figsize=(14, 6))
```

```
        # Длина ответа
        bars_len = axes[0].bar(labels, lengths, color=['purple', 'yellow'])
        axes[0].set_title('Длина ответа (символы)')
        axes[0].grid(True)
        for bar, length in zip(bars_len, lengths):
            yval = bar.get_height()
            axes[0].text(bar.get_x() + bar.get_width()/2, yval + 5, f"{yval}",
ha='center', va='bottom')

        # Время ответа
        bars_time = axes[1].bar(labels, times, color=['purple', 'yellow'])
        axes[1].set_title('Время ответа (сек)')
        axes[1].grid(True)
        for bar, t in zip(bars_time, times):
            yval = bar.get_height()
            axes[1].text(bar.get_x() + bar.get_width()/2, yval + 0.01, f"{yval:.2f}",
ha='center', va='bottom')

        plt.tight_layout()
        plt.savefig("comparison.png")
        plt.close()

        img = tk.PhotoImage(file="comparison.png")
        self.output_area.image_create(tk.END, image=img)
        self.output_area.img = img  # чтобы не удалить из памяти

    def export_history(self):
        df = pd.DataFrame(self.history)
```

```
        filename = filedialog.asksaveasfilename(defaultextension=".xlsx",
filetypes=[("Excel файл", "*.xlsx")])
        if filename:
            df.to_excel(filename, index=False)
            tk.messagebox.showinfo("Успех", f"История сохранена как {filename}")



# === Запуск приложения ===
if __name__ == "__main__":
    root = tk.Tk()
    app = LLMApp(root)
    root.mainloop()
```