Бронирование отелей и покупка авиабилетов при помощи LLM

1. Вступление

Современные технологии, такие как языковые модели с большим объемом данных (LLM), все больше проникают в сферу туристических услуг, меняя привычные способы бронирования и покупки. LLM - это мощные искусственные интеллекты, которые могут понимать естественный язык, обрабатывать большие объемы информации и предоставлять персонализированные рекомендации.

Использование LLM в процессе бронирования отелей и авиабилетов дает ряд преимуществ. Во-первых, эти модели могут быстро анализировать предпочтения пользователя, его бюджет и другие требования, чтобы предложить наиболее оптимальные варианты. Во-вторых, LLM способны обрабатывать естественные запросы, позволяя клиентам взаимодействовать с системой на привычном языке. Это значительно упрощает и ускоряет процесс планирования поездки.

Более того, LLM могут предоставлять дополнительную информацию, например, об инфраструктуре отелей, отзывах других путешественников или актуальных событиях в пункте назначения. Это помогает туристам принимать более обоснованные решения. В целом, применение LLM в сфере туризма открывает новые возможности для более персонализированного и эффективного обслуживания клиентов.

В данной работе будет использоваться два типа задач обработки естественного языка: классификация и NER (Named Entity Recognition). Для этих задач наиболее часто используют Encoder only transformer модели, например, BERT, но в данной работе я хочу продемонстрировать подходы, использующие Decoder only модели.

2. Данные

Для обучения и тестирования LLM на задачи классификации и NER необходимо иметь данные из специфичной области. В случае поставленной задачи по помощи в бронировании отелей и/или покупке авиабилетов нужно было собрать датасет, который бы содержал в себе записи сообщений клиентов в похожих сервисах с соответствующими именованными сущностями в них.

Поскольку было решено, что сервис должен работать на русском языке, то поэтому нужны было найти именно русскоязычные записи. К сожалению, в открытом доступе не удалось найти подобные.

Исходя из отсутствия данных, было решено сгенерировать их при помощи LLM. Для этого сначала были определены поля, которые для задачи NER модель должна уметь определять. В сообщениях о бронировании отелей нужно определять:

- город расположения отеля;
- название отеля;
- дата заселения в отель;
- количество гостей для проживания;
- количество дней пребывания в отеле.

А в сообщениях по покупке билетов:

- город отправления на самолете;
- город прибытия на самолете;
- дата отправления из города отправления;
- дата возвращения из города прибытия.

После этого необходимо было сгенерировать по тысяче записей в формате json для каждого типа задачи (бронирование, билеты). Для этого были найдены в открытом доступе списки отелей и городов, созданы функции по генерации случайных значений в определенных диапазонах для оставшихся полей в данных. С вероятность 20% некоторые из полей заполнялись пустыми данными, чтобы при генерации текста были более разнообразные.

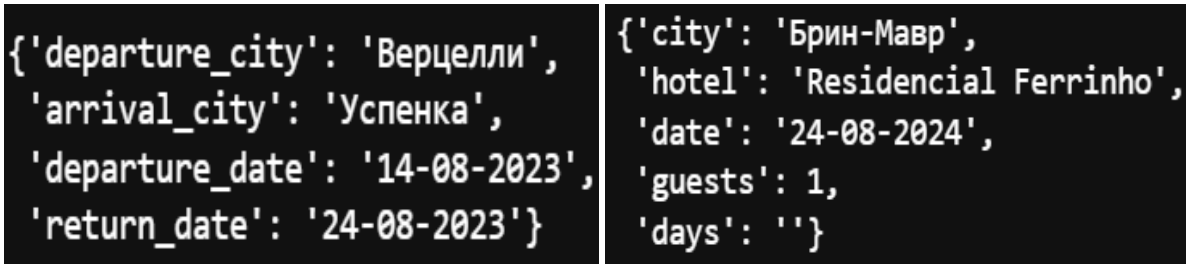

Рисунок 1. Пример сгенерированных json

Последним этапом был настроен промпт, которому на вход подавался json и описания его полей, после чего LLM должна была сгенерировать текст сообщения от клиента, используя данные из json. Для генерации использовалась модель «mistralai/Mistral-7B-Instruct-v0.2» и параметры семплирования: temperature = 1.2 и top_p = 0.5.

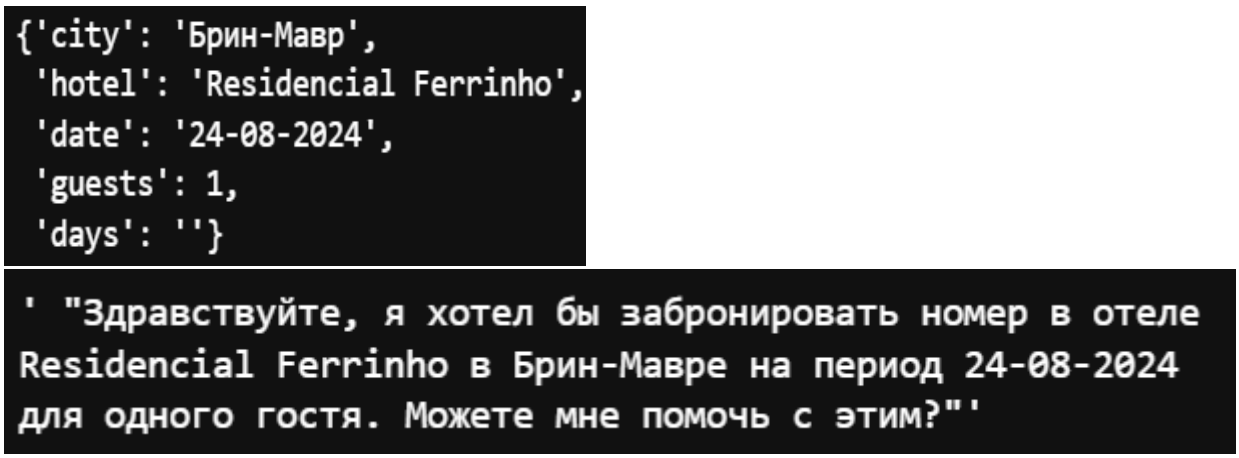

Рисунок 2. Пример сгенерированного текста

Таким образом получилось сгенерировать по одной тысяче записей для бронирования отелей и покупке авиабилетов.

3. Классификация

Следующим шагом необходимо было проверить способность LLM на классификацию входящего сообщения, чтобы в дальнейшем выбирать правильный промпт в зависимости от стоящей задачи.

Для решения задачи классификации с использованием LLM модели можно либо дообучить модель, либо использовать In-context learning, в данной работе используется второй вариант.

В тестировании участвовали две модели: “mistralai/Mistral-7B-Instruct-v0.2” и “mistralai/Mistral-7B-v0.1”. Первая модель доучена на следование инструкциям для вопросно-ответных систем, а вторая является базовой моделью, хранящей в себе основные знания о мире. Были выбраны именно эти модели для проверки необходимости дооубчения базовой модели на задачу классификации.

Использовались подходы zero-shot и one-shot learning после чего замерялась точность классификации моделей на всех сгенерированных данных.

	Mistral-7B-v0.1	Mistral-7B-Instruct-v0.2
Zero-shot «самолет»	0.998	0.996
One-shot «самолет»	1.000	0.999
Zero-shot «отель»	0.999	1.000
One-shot «отель»	0.997	1.000

Таблица 1. Результаты In-context learning для классификации

Видно, что обе модели справляются примерно одинаково, поэтому необходимость дообучать модель нет.

4. NER

4.1. Тестирование

Для тестирования использовалась только базовая модель Mistral-7B-Instruct-v0.2, поскольку предполагалось, что ее качество для задачи NER будет лучше базовой модели, т.к. она училась на следование инструкциям.

Был настроен промпт, в котором на вход подавалось описание атрибутов в json и текст, в котором нужно эти атрибуты найти. Точность замерялось для каждого атрибута отдельно путем сравнивания значений в сгенерируемом json и эталонном.

Сначала была замерена точность на всех имеющихся данных, чтобы определить способность модели к выполнению задачи NER. После чего точность замерили исключительно для тестовой выборки, которая составляет 20% от имеющихся данных, чтобы можно было определить прирост качества после дообучения.

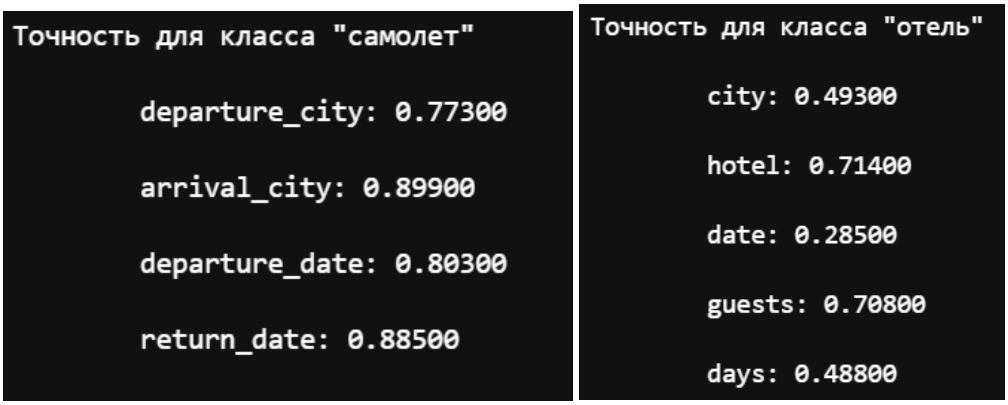

Рисунок 3. Точность NER на всех данных

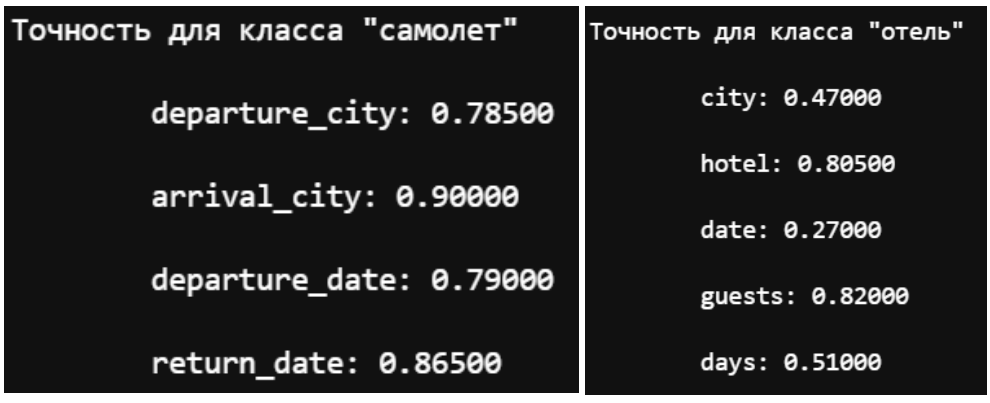

Рисунок 4. Точность NER на тестовых данных

Можно заметить, что модель справляется не идеально с поставленной задачей, проанализировав данные в ходе тестирования можно сделать следующие выводы:

- в классе «самолет» модель периодически путает город отправления и город прибытия;
- в классе «отель» название города попадает в название отеля и наоборот;
- в классе «отель» плохо определятся дата заезда и количество гостей;

В целом, есть атрибуты, поиск качества которых можно улучшить в ходе процесса дообучения.

4.2. Обучение

В дообучении участвовала базовая модель Mistral-7B-v0.1, чтобы продемонстрировать улучшение качества в задачи NER с более качественной моделью Mistral-7B-Instruct-v0.2.

Был создан тренировочной датасет, который составляет 80% от имеющихся данных, настроен промпт для задачи NER.

Для обучения использовался LoRA адаптер со следующими параметрами:

- r = 16;
- lora_alpha = 32
- lora_dropout = 0.05

Были выбраны слои из последнего блока декодера и выходной линейный слой. Общее кол-во тренируемых параметров относительно всех параметров модели составило ~0.03%.

Параметры тренировки:

- batch_size=4;
- max_steps=200;
- lerning_rate=2.5e-5;
- optimizer=adamw_bnb_8bit

Требуемое кол-во VRAM – 35Гб.

Была выбрана только одна эпоха, для предотвращения переобучения модели.

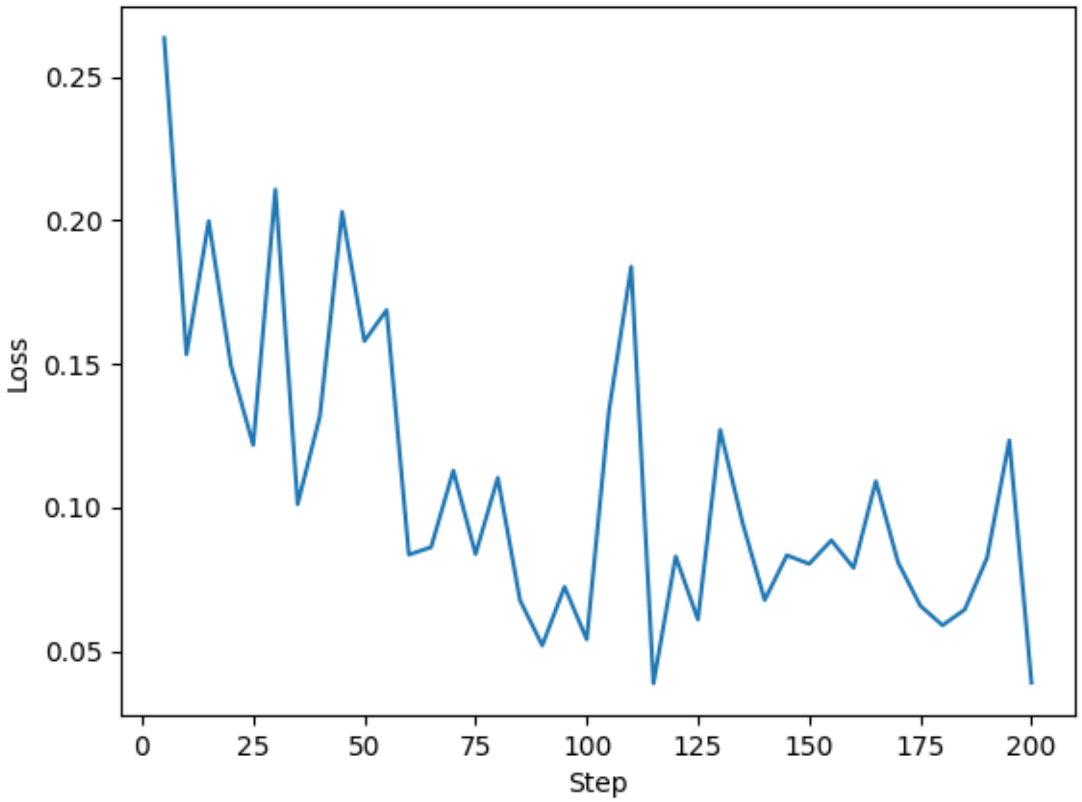

Рисунок 5. График обучения

После обучения LoRA адаптер был смержен с основной моделью и сохранен в отельную папку, чтобы новую модель можно было использовать в фреймворке vLLM.

Протестировав новую модель, можно заметить значительное улучшение качества для определенных атрибутов, теперь модель имеет примерно одинаковую точность для каждого из них. Также точность классификации не пострадала.

Атрибут	Точность до обучения, %	Точность после обучения, %	Изменение точности, %
«Самолет». Город отправления	0.785	0.865	+0.08
«Самолет». Город прибытия	0.9	0.895	-0.005
«Самолет». Дата отправления	0.79	0.79	0
«Самолет». Дата возвращения	0.865	0.815	-0.05
«Отель». Город отеля	0.47	0.585	+0.115

«Отель». Название отеля	0.805	0.915	+0.110
«Отель». Дата заселения	0.27	0.865	+0.595
«Отель». Количество гостей	0.82	0.96	+0.14
«Отель». Количество дней проживания	0.51	0.6	+0.09

Таблица 2. Изменение точности для задачи NER

Таким образом, после дообучение имеется модель, которая способна решить поставленные задачи.

5. Архитектура

Вокруг обученной модели был построен сервис для тестирования функционала модели.

Разработанный сервис состоит из двух составляющих: API сервис и веб-интерфейс. В качестве фреймворка для API был выбран FastAPI, а для веб-интерфейса – Streamlit.

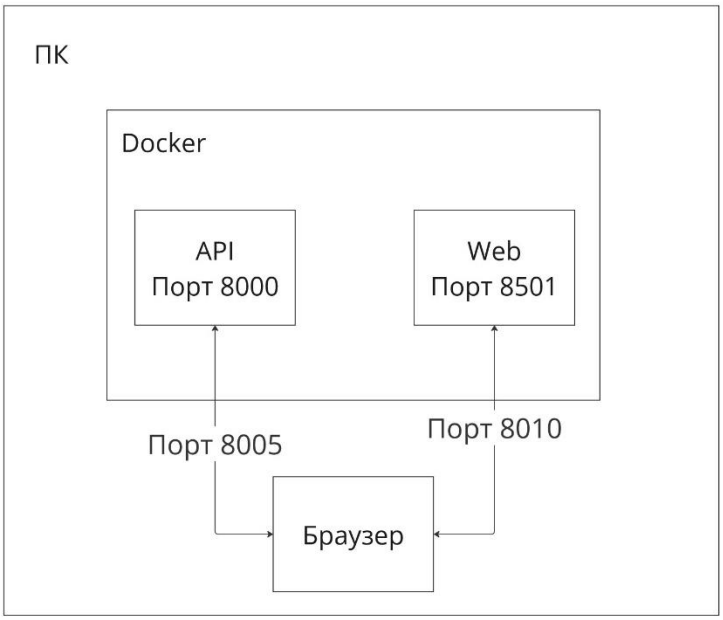

Рисунок 6. Архитектура решения

Веб-интерфейс выполнен в виде чата со стриминговым получением данных.

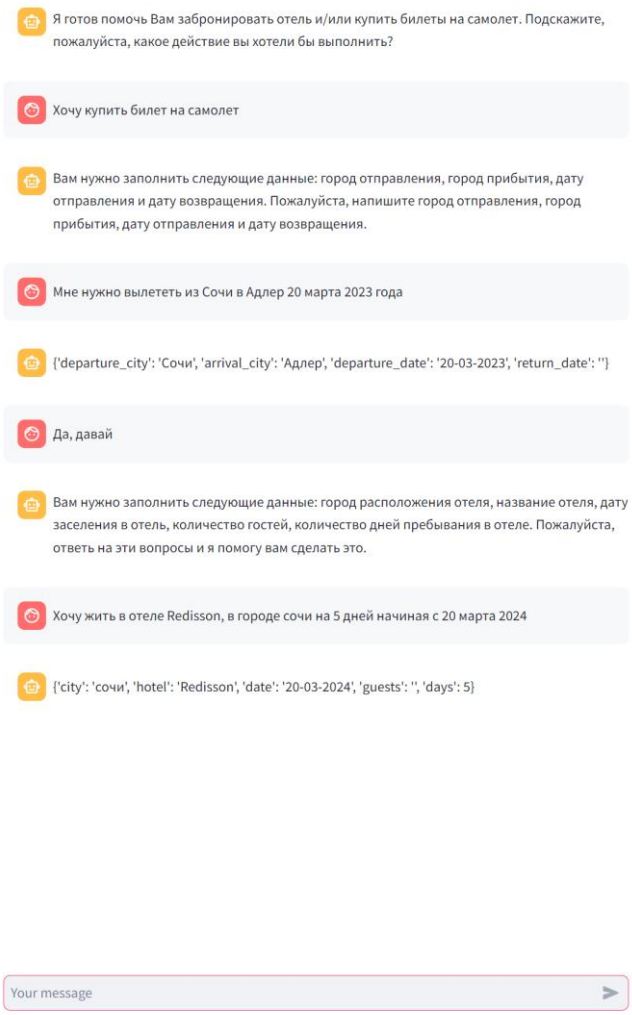

Рисунок 7. Веб-интерфейс

При получении json после NER он передается в mock API из контейнера фронта.

Функциональность создана таким способом, что пользователь пишет о своей потребности и чат-бот помогает с заполнением полей для поиска информации. В одном чате пользователь может получить услугу как по бронированию отеля, так и по покупке билета при желании. После выполнения обеих функций чат становится недоступным и для повторного запроса надо обновить страницу.

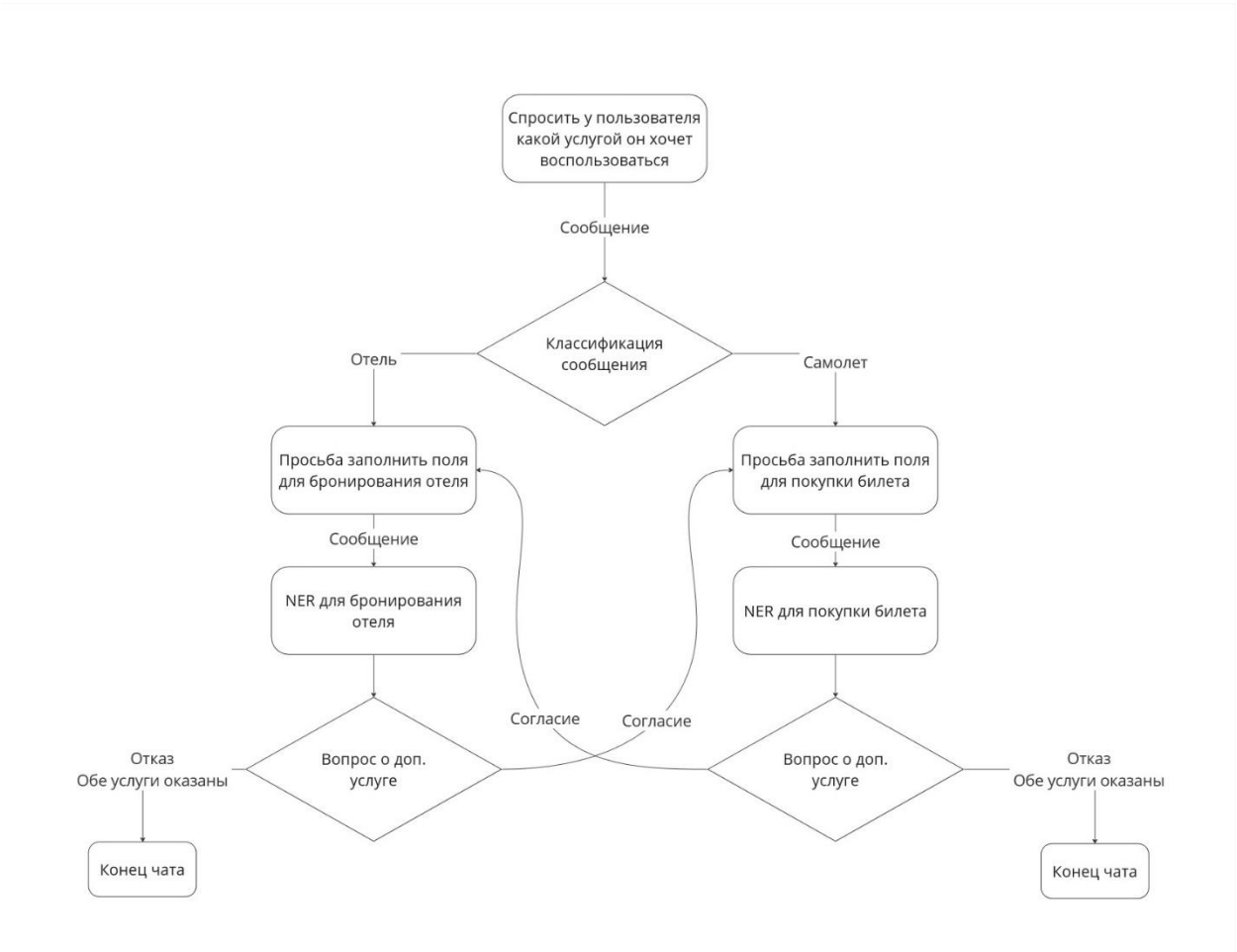

Рисунок 8. Дерево решений общения в чате

В данном сервисе LLM классифицирует как классы запроса, так и ответ пользователя о доп. услуге (согласен/не согласен).

6. Заключение

В ходе работы удалось проверить качество LLM моделей для задач классификации и NER, дообучить модель и построить демо стенд для модели. Имеются некоторые архитектурные решения, которые не получилось реализовать, например, более качественная диалоговая система и корректировка промптов.

Для дальнейшего исследования нужно протестировать BERT-like модели в сравнении с получившимся результатом для LLM, а также провести эксперименты с параметрами семплирования и промптами. Использование истории чата, тоже может принести свои плоды.

В заключение, хочется сказать, что финальное качество модели для поставленных задач довольно высокое, но еще есть огромное поле для тестирования и улучшения данной работы.