Проект по Продвинутым Методам Машинного Обучения

Уколов Степан, Сиворакша Тимофей, Алиев Элвин, Валерий Ходжаев

18.12.2024

Содержание

1 Введение

Пользователь

сгенерируй смешную картинку для тад проекта по поиску домов для аренды

ChatGPT

Бедные орги каждый год долго-долго выбирают дома на посвят(Мы решили, что так больше продолжаться не может: ~~отменяем посвят~~ делаем супер-пупер RAG систему с возможностью быстрого и удобного подбора релевантных объявлений! Как говорится «Лофты размыли ценность посвята», пора решать эту проблему!

Хочеца введение ещё расписать, картинок по вставлять, но так уже лень... Тем более во время написания введения в гжк упал инет(

Проект посвящён обработке данных по объявлениям аренды загородных домов с платформы ЦИАН. Как говорится, начинали с простого анализа, а закончили риелторским стартапом. Но обо всём по порядку!

P.S.

В этой пдфке много разных гиперссылок на “жирном тексте” (не на заголовках) => не стесняемся, наводим курсор, пробуем тыкать

2 Сбор данных

2.1 Почему именно Циан?

Изначально предполагалось парсить Авито, но тщетные N часов попыток дали понять, что Авито, скорее всего, настолько устали от всяких парсеров/скрэтчеров, что

1. Понавтыкали своих cloudlfare, который еще и умеет смотреть на IP юзера, а не на имя агента в headers, да так, что всяких базовые библиотеки питона для обхода cloudlfare не лечат траблы.
2. Повставляли на страницы элементы, которые грузят javascript контент, из-за чего нужны умные библиотеки для запросов, которые умеют рендерить такие штуки (eg. HTMLrequests, Selenium)
3. Придумали сложную генерацию ссылок со вставкой параметров, что глаза вытекают
4. Спрятали все обращения к своим API

В то время как Циан оставили кнопочку на сайте <Скачать в *Excel*>, но увы, там стоит ограничение на 200 домов, к тому же кнопку можно нажимать раз в сутки. Мы же хотим получить базу побольше. Как и Авито, Циан генерирует ссылку на страницу с домами в зависимости от введённых параметров на основной странице. Так как мы не хотим проводить посвят в квартире или где-то в Мурманске и, к сожалению, не будем брать дом на срок более одних суток, вбиваем нужные параметры, а сгенеренную ссылку берем за основную. Снова приятный сюрприз, ссылки страниц домов оставлены на самой странице внутри одних и тех же тегов. Да и сами ссылки страниц домов созданы по понятному принципу: cian + имя ареала (eg. odintsovo) + внутренний id сущности в базе Циана. Более того, Циан предлагает пользователю только активные объявления, в отличие от ~~мат~~ дурацкого Авито.

2.2 Парсер Циан

Стэк джентльмена:

```
from bs4 import BeautifulSoup
import requests
import pandas as pd
```

В целом Циан - френдли сервис для парсинга. Даже на тот факт, что юзер отправляет слишком много запросов, он отвечает не 427 статус кодом, разрывая конекшн, как большинство сайтов, а 429 кодом, в котором просит снизить частоту. Как мило.

Резюмируя, каков алгоритм парсера:

1. Переходит по основной ссылке со списком домов:

```
def get_houses_links(self):
        houses_links = []

        logger.info('Getting houses links...')

        for page_number in tqdm(range(1,
            self.max_pages_cnt+1), file=sys.stdout):
            response = self.do_get_request(url
            =self.main_link.format(page_number=page_number))
            soup = BeautifulSoup(response.text, 'html.parser')
            pred = soup.find_all('a',
                href=re.compile("suburban"),
                class_='_93444fe79c--link--VtWj6')
            loop_list = [tag.get('href') for tag in pred]
            houses_links += loop_list

        houses_links = list(set(houses_links))

        return houses_links
```

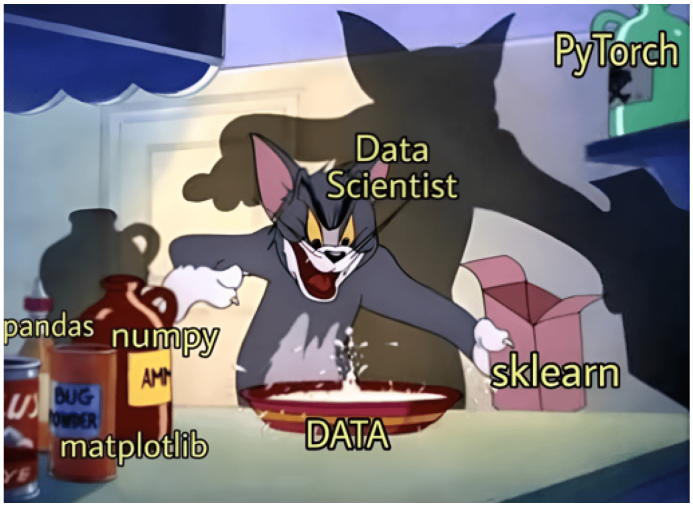
2. Собирает все ссылки. Делается это примитивным увелечением параметра p в ссылке на 1, для того, чтобы переключаться между частями списка. Даже тут Циан не стал изворачиваться с технологией динамической подгрузки контента (AJAX). Большое спасибо команде веб-разработки.

3. Парсер пробегается по ссылкам страниц домов и забирает из html тегов нужные нам данные.

4. На случай, если сервера Циан ругаются на наше количестве запросов, срабатывает retry функция, которая ненадолго останавливает парсер, чтобы выждать нужное время, когда сервер снова даст возможность делать запросы.

Финальные замечания. Хоть это и не общее решение, то есть, универсально спарсить любую инфу с Циан не получится. Но для нашей задачи такого и не требуется. А если философствовать, то в настоящем бизнес-решении верить, что парсер - полноценный сурс для ETL пайплайнов, идея мягко говоря бредовая. Были бы ресурсы - купили бы доступ к API.

3 Препроцессинг данных



Общая схема работы

1. **Извлечение корректных цен аренды** из текстовых описаний.
2. **Моделирование отсутствующих цен** с использованием крупной языковой модели **Qwen2.5 72b-instruct-q4_0**.
3. **Добавление географических координат** для визуализации на карте.
4. **Фильтрация данных** для поиска **идеальных домов для посвята**.

3.1 Первичный препроцессинг данных: готовим поле для работы

Перед тем как натравить модели на текст, мы провели первичную чистку данных. Иначе наш датасет выглядел бы как объявление: “дом, не дом, цена, не цена, ну вы поняли...”.

Этапы первичного препроцессинга:

1. **Удаление дубликатов:**

   На ЦИАН дома иногда размножаются, как котята. Дубликаты были безжалостно удалены:

2. **Заполнение пропусков:**

   Строки без адреса и описаний? Это как дом без крыши — непригодно для работы. Добавили заглушки:

```
df['Address'] = df['Address'].fillna('
                ')
df['Description'] = df['Description'].fillna('There is no
    description')
```

3. **Удаление аномалий:**

   Цена в 500 рублей? Это что, палатка в лесу? Оставили только разумные диапазоны:

```
df = df[(df['price'] > 1000) & (df['price'] < 10000000)]
```

3.2 Извлечение корректных цен с использованием Qwen2.5 72b

Цены на ЦИАН — это отдельный вид искусства. Указано “от 10 000 рублей”, а по факту — “ещё 50 000 за бассейн и два полотенца”. Поэтому был разработан гибридный подход:

1. **Извлечение цен из текста**

   Мы написали функцию extract_prices_from_text, которая находит все 4-6 значные числа в текстах:

```
def extract_prices_from_text(text):
    prices = re.findall(r'\b\d{4,6}\b', text.replace(' ', ''))
    return [int(price) for price in prices] if prices else
        None
```

   Пример работы: “Аренда дома за 50 000 рублей в сутки. С бассейном — 70 000.” Функция возвращает: [50000, 70000].

2. **Оценка цен с Qwen2.5 72b-instruct-q4_0**

   Если функция не находит цен в тексте, то мы говорим Qwen2.5: “Думай сама!”.

   Как это работает:

   Генерируется промпт с описанием и площадью дома:

   prompt = f"Учитывая описание дома: "description"и площадь area м^2: - Если в описании указаны цены, рассчитайте среднюю стоимость аренды дома для выходного дня. - Если цены не указаны, оцените стоимость аренды, учитывая площадь, расположение, состояние и характеристики. Ответ должен быть только числом (стоимость аренды в день, руб.). "

predicted_price = model(prompt)

Ответ модели проверяется и парсится. Если Qwen вдруг решит поговорить, а не ответить числом, мы вежливо округлим ей мысли до ближайшей 1000 рублей.

3. **Корректировка цен** После извлечения или предсказания цен мы добавили логику:

   Маленькие дома (< 100 м^2) $\rightarrow$ минимальная цена 10000 руб.

   Средние дома (100–300 м^2) $\rightarrow$ минимальная цена 25000 руб.

   Большие дома (> 300 м^2) $\rightarrow$ минимальная цена 50000 руб.

4. **Фильтрация данных** для поиска **идеальных домов для посвята**.

3.3 Геокодирование адресов

Следующим этапом стало геокодирование адресов с помощью Яндекс Геокодера. Как говорится, “написать адрес — одно, а найти дом — совсем другое”.

Шаги геокодирования:

1. **Очистка адресов:**

   Убираем текст в скобках, нормализуем сокращения:

   “д.” $\rightarrow$ “дом”, “ш.” $\rightarrow$ “шоссе”.

2. **Запросы к Яндекс API:**

   Для каждого адреса отправляется запрос на геокодирование, и в ответе мы получаем широту и долготу

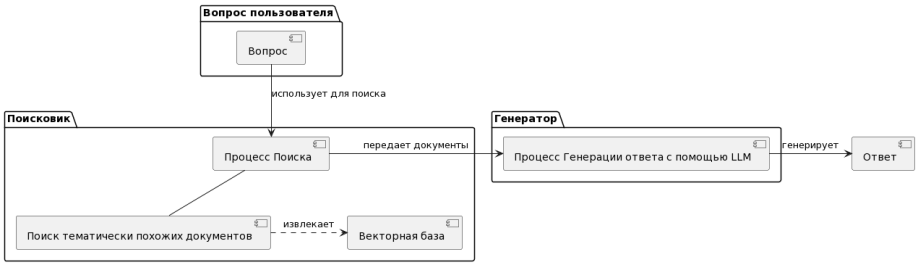
```
df["Latitude"], df["Longitude"] =
    zip(*df["Address"].apply(lambda x: get_coordinates(x,
    api_key)))
```

4 RAG

Если, открывая холодильник, вы еще не слышали из него про RAG - то наверняка скоро услышите.

RAG — это Retrieval-Augmented Generation (известный так же под позывным RAG). По сути, идея простая как пять копеек - взять существующую хорошую модель (тот же OpenAI), и прикрутить ей сбоку поиск по информации компании. Модель все еще мало что про вашу компанию знает, но теперь у нее есть где поглядеть. Это не так эффективно, как если бы она знала, но достаточно для большинства задач.

Базово RAG выглядит следующим образом:



Ликбез окончен, можно приступать к нашему решению!

Общая схема работы

1. **Загрузка данных из CSV-файла.**
2. **Создание векторных эмбеддингов для текстов и индекса поиска.**
3. **Поиск релевантных текстов в индексе на основе запроса.**
4. **Настройка языковой модели с параметрами и контекстом.**
5. **Генерация ответа моделью, включая найденные данные.**

4.1 Загрузка данных из CSV-файла.

```
file_path = 'house_dataset.csv'
loader = CSVLoader(file_path=file_path)
docs = loader.load_and_split()
```

- **CSVLoader**: Загружает CSV-файл и конвертирует строки в документы (каждая строка CSV становится отдельным текстовым документом).
- **load_and_split**: Загружает файл и делит данные на документы. Эти документы затем будут использоваться для построения поискового индекса.

4.2 Создание векторных эмбеддингов для текстов и индекса поиска.

```
class SentenceTransformerEmbeddings:
    def __init__(self,
        model_name="intfloat/multilingual-e5-large-instruct"):
        self.model = SentenceTransformer(model_name)

    def embed_query(self, query):
        return self.model.encode([query],
            convert_to_tensor=False)[0]

    def embed_documents(self, docs):
        return self.model.encode(docs, convert_to_tensor=False)
```

- **SentenceTransformer**: Это библиотека, созданная для работы с трансформерами (Transformer) с целью преобразования текста в числовые векторы (эмбеддинги). Используется модель

  intfloat/multilingual-e5-large-instruct, которая поддерживает несколько языков.
- Методы:
  - **embed_query(query)**: Преобразует строку-запрос в вектор.
  - **embed_documents(docs)**: Преобразует массив текстов в массив векторов.

Эти эмбеддинги нужны, чтобы сравнивать запросы с текстами из базы.

Что это за модель?

Есть такая штука **MTEB (Massive Text Embedding Benchmark** - это платформа для оценки моделей текстовых эмбеддингов по множеству задач (поиск, классификация, кластеризация и т.д.) на разных языках. Она позволяет объективно сравнить качество преобразования текста в числовые векторы. Чуть-чуть тыкаем, выбираем нужные для нас параметры и берём лучшую модель под нашу задачу!

Берём **эту**, она инициализирована на основе xlm-roberta-large и дополнительно обучена на смеси мультиязычных наборов данных. Ну и она лидирует во многих частях нашего лидерборда, в частности на задачах с русскоязычными текстами.

4.3 Поиск релевантных текстов в индексе на основе запроса.

```
embedding_dim = len(embeddings.embed_query(" "))
index = faiss.IndexFlatL2(embedding_dim)
vector_store = FAISS(
    embedding_function=embeddings.embed_query,
    index=index,
    docstore=InMemoryDocstore(),
    index_to_docstore_id={}
)
```

- **FAISS**: Библиотека для поиска ближайших соседей по векторным представлениям. Это основа для поиска релевантных документов.
- **IndexFlatL2**: Используется индекс с метрикой L2 (евклидово расстояние) для поиска похожих векторов.
- **vector_store**: Хранилище, объединяющее:
  - Векторный индекс.
  - Функцию генерации эмбеддингов **embed_query**.
  - Документы (**InMemoryDocstore** хранит текстовые документы в памяти).

Что такое **FAISS**?

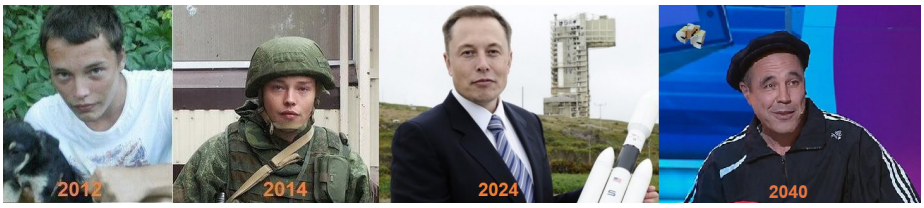
Facebook AI Similarity Search[1] – разработка команды Facebook AI Research для быстрого поиска ближайших соседей и кластеризации в векторном пространстве. Высокая скорость поиска позволяет работать с очень большими данными – до нескольких миллиардов векторов

Одним из главных преимуществ FAISS является инвертированный векторный индекс (IVF), который ускоряет поиск даже при работе с большими объёмами данных.

[1] Осуждаем, у нас такое нельзя, больше не будем. В следующий раз честно будет QDrant

FAISS также поддерживает дополнительные методы, такие как квантование продукта (PQ) и графы ближайших соседей (HNSW), которые позволяют проводить поиск ближайших векторов в наборах данных, содержащих миллиарды векторных представлений. Эти технологии обеспечивают возможность обработки больших массивов данных, таких как базы изображений, текстов или аудиофайлов.

Вообще говоря, FAISS открывает огромное поле для реализации каких-либо творческих идей. Например, по тому же принципу векторной близости похожих лиц можно было бы строить пути от одного лица к другому. Или в крайнем случае сделать из FAISS фабрику по производству подобных мемов:



4.4 Настройка языковой модели с параметрами и контекстом.

```
model_name = "ai-sage/GigaChat-20B-A3B-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
llm = LLM(model=model_name, trust_remote_code=True,
    tensor_parallel_size=2, max_model_len=24000)
sampling_params = SamplingParams(temperature=0.3, max_tokens=1000)
```

Пора бы и LLM-ку выбрать

Будем смотреть по бенчмаркам:

- **MERA**
- **SLAVA**

Изначально

Выбор пал на Квены. Мы хотим опенсорсную мультиязычную модель. Денег на апи яндекса и гигачата нет (хотя они и предоставляют пару бесплатных токенов). Новые моедли от Т-банка - те же дообученые квены, но они с HF, а мы хотим всё поднять на ollama. Короче крутое китайское семейство, которое легко поднять отовсюду!

Однако от этой идеи пришлось отказаться так как:

13 декабря Сбер выкладывает GigaChat Lite в открытый доступ

GigaChat-20B-A3B-instruct - Диалоговая модель из семейства моделей GigaChat, основная на GigaChat-20B-A3B-base. Поддерживает контекст в 131 тысячу токенов.

Почему выбрали именно `GigaChat-20B-A3B-instruct`?

1. **Размер модели (20b):**
    - Достаточно мощная для сложных запросов, но не слишком большая, чтобы требовать дорогостоящих ресурсов. Модели меньшего размера (например, 7b) чуть менее точны, так как имеют меньше параметров для обучения сложных закономерностей. С другой стороны, модели с параметрами >25b требуют больше вычислительных ресурсов, что делает их сложными для использования в реальном времени.
2. **instruct:**
    - Идеально подходит для задач, где требуется чёткий и прямой ответ на запрос.

Параметры модели сильно не трогаем, главное температурку по-меньше, больше будет полагаться на полученную информацию, меньше выдумывать!

Изначально модель выбиралась по выложенным лидербордам с бенчмарков, но новый гигачат протестить нигде официально не успели. Неофициально она была протестирована на славе, где показала очень хорошие результаты, догнав 72-миллиардный квен. + её оказалось очень просто запустить через vllm (блин, придётся дальше написать, что это и почему круче ollama), => выбор пал на неё.

vLLM

vLLM — это высокопроизводительная система для ускоренного развертывания и обслуживания больших языковых моделей (LLM). Она разработана с акцентом на максимальную производительность, низкие задержки и высокую пропускную способность. vLLM особенно полезна для задач инференса, где важна скорость генерации текстов от модели и одновременная обработка множества запросов.

Чем vLLM круче Ollama?

1. **Производительность:**
   vLLM обеспечивает максимальную скорость и высокую пропускную способность за счет оптимизации PagedAttention и работы на GPU. Ollama фокусируется на простом и удобном запуске моделей, но уступает в производительности на больших нагрузках.

2. **Интеграция с батчингом:**
   vLLM позволяет объединять множество запросов в батчи, эффективно используя ресурсы GPU. Ollama менее ориентирована на батчинг и массовый инференс.

3. **Гибкость моделей:**
   Хотя Ollama также поддерживает разные модели, vLLM предоставляет широкие возможности по кастомизации и интеграции моделей, что делает её более универсальной.

Грубо говоря vLLM - Assembler в мире систем развертывания и обслуживания LLM, а Ollama тот же самый python.

4.5 Генерация ответа моделью, включая найденные данные.

```
def create_vllm_chain(retriever, tokenizer, llm, sampling_params, system_message):
    def answer_question(input_text):

        retrieved_docs = retriever.get_relevant_documents(input_text, k=10)
        context = "\n\n".join([doc.page_content for doc in retrieved_docs])

        messages = [
            {"role": "system", "content": system_message.content},
            {"role": "user", "content": f"                :
                {context}\n\ n            : {input_text}"}
        ]

        prompt_token_ids = tokenizer.apply_chat_template(messages, add_generation_prompt=True)

        outputs = llm.generate(prompt_token_ids=[prompt_token_ids], sampling_params=sampling_params)

        generated_text = outputs[0].outputs[0].text
        return generated_text

    return answer_question
```

- `retriever.get_relevant_documents(input_text, k=10)`:

  Находит 10 самых релевантных документов для запроса.

- `context`:

  Создаёт строку из найденных документов.

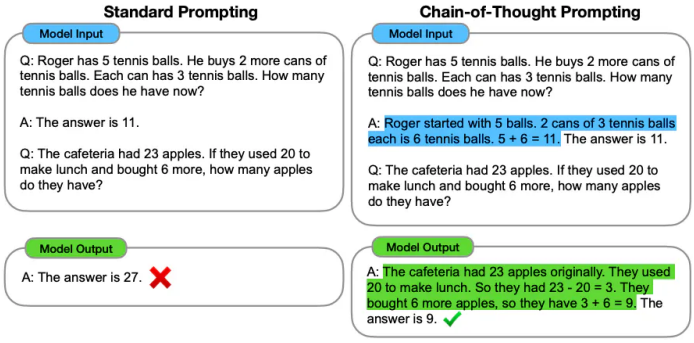
- `model.invoke(messages)`:

  Передаёт модель сообщение с контекстом и запросом.

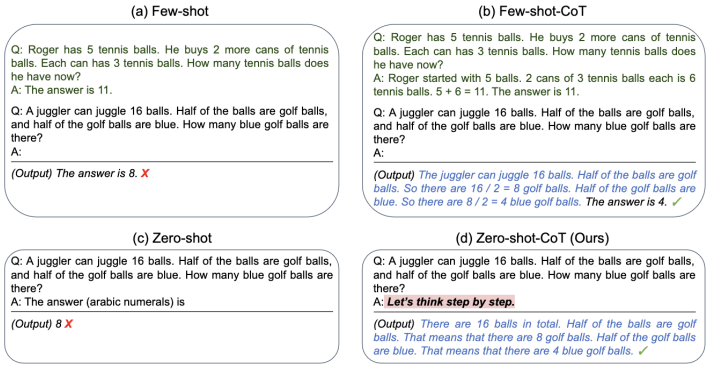
- **Возвращает ответ модели, либо сообщение об ошибке.**

Prompt engineering.

system_message? Можно бесконечно долго заниматься промт тюнингом, но мы сильно не мучались, а сгенили себе хороший промт вот **тут**. Поправили так, чтобы у нас был CoT (Chain-of-Thought) и few-shotы.



Введенная в Wei et al. (2022) техника формулировки промптов "цепочка мыслей"(CoT) позволяет выполнять сложные рассуждения с помощью промежуточных шагов рассуждения. Её комбинация с few-shot позволяет получить лучшие результаты в более сложных задачах, требующих рассуждения перед ответом.


(a) Few-shot
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:
(Output) The answer is 8. ✗
(b) Few-shot-CoT
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:
(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓
(c) Zero-shot
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is
(Output) 8 ✗
(d) Zero-shot-CoT (Ours)
Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: Let's think step by step.
(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

5 Веб-сервис

Ох, блин...

Есть такая штука как Flask - гибкий и интуитивно понятный способ создания веб-приложений. Основой Flask является маршрутизация запросов и генерация ответов. Но... Чтобы было красиво надо разбираться с html-разметкой, а мы этого не хотим, но красиво хотим)

=> streamlit/gradio

5.1 Streamlit

Streamlit - это фрэймворк Python с открытым исходным кодом для AI / ML специалистов, который позволяет создавать динамические приложения для обработки данных всего с помощью нескольких строк кода. Короче красотулька за 5 минут (почти) чтения документация!

Мы взяли всю нашу RAG-систему и интегрировали в интерфейс streamlita:

1. Предварительно поднимаем vLLM сервер:

```
vllm serve ai-sage/GigaChat-20B-A3B-instruct \
  --disable-log-requests \
  --trust_remote_code \
  --dtype bfloat16 \
  --max-seq-len 20000 \
  --tensor-parallel-size 2
```

   tensor-parallel-size 2 - да у нас 2 видюхи A6000, просьба никому не завидовать

2. Забираем наш промт:

```
prompt_path = Path("data/system_prompt.txt")

with prompt_path.open("r", encoding="utf-8") as file:
    prompt_content = file.read()
```

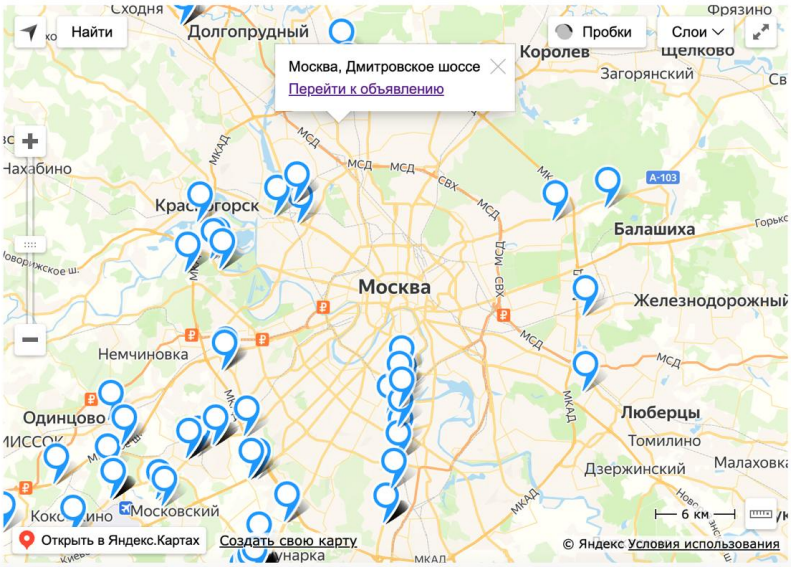
3. Настраиваем окошко с чат ботом:

```
for message in st.session_state.messages:
    with st.chat_message(message["role"]):
        st.markdown(message["content"])

if prompt := st.chat_input("
                           :"):

    st.session_state.messages.append({"role": "user",
        "content": prompt})

    with st.chat_message("user"):
        st.markdown(prompt)

    response = answer_question(prompt)
    st.session_state.messages.append({"role": "assistant",
        "content": response})

    with st.chat_message("assistant"):
        st.markdown(response)
```

4. Также мы использовали **API Яндекс Геокодера**, чтобы получить широту и долготу, а потом с помощью дополнительной html-разметки вывести всё красиво на яндекс карты:

```
def get_coordinates(address, api_key):
    url = f"https://geocode-maps.yandex.ru/1.x/"
    params = {
        "apikey": api_key,
        "geocode": address,
        "format": "json",
    }
    try:
        response = requests.get(url, params=params)
        response.raise_for_status()
        result = response.json()

        pos = result["response"]["GeoObjectCollection"]
        ["featureMember"][0]["GeoObject"]["Point"]["pos"]
        lon, lat = pos.split(" ")
        return float(lat), float(lon)
    except (IndexError, KeyError):
        return None, None
    except requests.RequestException as e:
        print({e})
        return None, None
```

Яндекс.Карты с данными из DataFrame и ссылками



5. Чтобы всё это уместилось в одном месте сделали page_config для размещени всей этой красоты на нескольких страницах!

```python
import streamlit as st

main_page = st.Page("main.py", title="                "
)
bot_page = st.Page("bot.py", title="       -       "
)
maps_page = st.Page("maps.py", title="           "
)

pg = st.navigation(
    {
            "Main": [main_page, bot_page, maps_page],
        }
)
st.set_page_config(page_title="House bot"
)
pg.run()
```

Весь код стримлита можно глянуть на **гите**, тут показали ключевые моменты!

5.2 Docker/poetry

Я искренне надеюсь, что Тимоха+Я сделаем это, но пока сильно под вопросом.

upd: надеюсь, что сделаю это (poetry) до защиты проекта и залью на гит, пока что верится в это с трудом. Никита, если ты это читаешь, то знай, что возможно на гите код который можно запустить через poetry.

Веб сервис будет работать по ссылке внизу только на время защиты(Ну или по требованию высшего руководства (Никиты). Можно будет зайти и потыкаться!

6 Заключение

Всем спасибо, всем пока! Надеюсь кто-нибудь до этого дочитает, иначе зачем всё это? Супер открыты к фидбэку по этой пдфке (ошибки в тексте, формулировках, различные несостыковки)

Список использованных ресурсов с гиперссылками:

- **FAISS**
- **RAG**
- **MTEB**
- **MERA**
- **Prompt engineering**
- **LLM**
- **Streamlit**
- **GigaChat**

Наша гордость:

- **GitHub**
- **Веб-сервис**

НАША КОМАНДА

Никнейм	**Задачи**
Валерий Ходжаев	Сбор данных
Тимофей Сиворакша	Препроцессинг
Уколов Степан	RAG
Алиев Элвин	Веб-сервис

Сэнкью фор ёр этеншон!