Отчет по архитектуре и системному дизайну
Проект: GeoCLIP FastAPI Service

Команда: Панасюк Михаил Михайлович

Май 2025

Аннотация

В данном отчете представлено проектирование системы GeoCLIP FastAPI Service, предназначенной для решения задачи геолокации по изображениям и поиска близлежащих достопримечательностей. Рассмотрены архитектура, оценка данных, выбор модели, системный дизайн, методы оценки качества сгенерированных предсказаний, а также этические и социальные аспекты проекта.

Содержание

1 Введение

1.1 Цель проекта

Цель проекта GeoCLIP FastAPI Service — создать прототип AI-системы, способной по загруженному изображению определить предполагаемые GPS-координаты места и найти ближайшие достопримечательности из заранее подготовленной базы данных. Решение использует мультимодальную модель GeoCLIP, дообученную для задачи геолокации, а также веб-сервис на FastAPI и интерактивный интерфейс на Streamlit+Folium.

1.2 Проблема и ценность

Проблема: пользователь загружает случайное фото местности или достопримечательности, но не знает точного географического положения кадра. Ручной поиск по карте и описаниям занимает много времени и не всегда дает точный результат.

Ценность:

- **Для туристов:** быстро определить, где сделано фото, и получить информацию о близлежащих объектах.
- **Для маркетинга и недвижимости:** анализ геолокаций популярных объектов на основе фото, планирование рекламных кампаний.
- **Для образовательных проектов:** демонстрация возможностей мультимодальных моделей (CV+геоданные).

2 Проектирование системы

2.1 Обзор архитектуры

В системе выделены следующие основные компоненты:

- **Клиентский интерфейс:** веб-приложение на Streamlit. Пользователь загружает изображение или задает координаты вручную, получает результаты.
- **API-сервис:** FastAPI, exposes три эндпоинта:
  - `GET /health` — проверка работоспособности;
  - `POST /predict/coords` — предсказание координат по изображению;
  - `POST /search/nearby` — поиск ближайших достопримечательностей по изображению;
  - `GET /examples/nearby` — поиск ближайших по введенным координатам.
- **Модель GeoCLIP:** PyTorch-модель, дообученная на паре {изображение, GPS}. Загрузка модели происходит при старте API, инференс происходит в памяти.
- **База данных PostgreSQL:** таблица `images` со столбцами `id, name, lat, lon, url`. Инициализируется скриптом `docker/db/init.sql` при старте контейнера.
- **Пакет для работы с БД:** SQLAlchemy ORM (файл `app/db.py`), функции доступа к данным (`app/database.py`), включая функцию поиска ближайших объектов по формуле Haversine.

- **Docker Compose:** поднимает три сервиса: `db` (PostgreSQL), `api` (FastAPI+Uvicorn), `frontend` (Streamlit).

Рис. 1: Упрощенная схема архитектуры системы

2.2 Взаимодействие компонентов

1. Пользователь обращается к фронтенду (Streamlit).
2. Frontend отправляет запрос в API (`/predict/coords` или `/search/nearby` или `/examples/nearby`
3. FastAPI контроллер сохраняет загруженное изображение во временный файл, вызывает функцию инференса `predict_topk(model, image_path, top_k)`.
4. Модель GeoCLIP (PyTorch) возвращает список координат с вероятностями.
5. В случае `predict/coords` этот список возвращается клиенту.
6. В случае `search/nearby`, первые координаты (центр) подаются в функцию `search_nearby(cent` `radius_km, db)`, использующую Haversine для поиска строк таблицы `images` в указанном радиусе.
7. API возвращает JSON с `center` и списком `matches` (id, name, lat, lon, url, distance_km).
8. Frontend получает ответ, строит таблицу и интерактивную карту (Folium).

2.3 Используемые технологии

- **Язык**: Python 3.10
- **Web-фреймворк**: FastAPI + Uvicorn
- **Модель**: PyTorch + библиотека `geoclip`
- **ORM**: SQLAlchemy, `psycopg2-binary`
- **База данных**: PostgreSQL 15
- **Маппинг на клиенте**: Streamlit + Folium + `streamlit-folium`
- **Контейнеризация**: Docker, Docker Compose

3 Оценка данных

3.1 Источники данных

- Набор данных для дообучения GeoCLIP (готовый вес модели), содержащий пары {изображение, GPS-координаты}. Обычно публичные датасеты: YFCC100M, IM2GPS, Google Landmarks Dataset, но в нашем прототипе используем предобученную модель GeoCLIP без дополнительного обучения.
- Встроенная база тестовых точек (`METADATA` в `app/database.py` и в PostgreSQL) — 23 известных достопримечательностей (Эйфелева башня, Биг-Бен, Храм Христа Спасителя, Эрмитаж и т.д.).

3.2 Качество и объем данных

- **Модельные данные:** GeoCLIP была дообучена на тысячах изображений с точными метками GPS.
- **База тестовых точек:** всего 23 строки (объем малый, но достаточный для демо).
- **Проблемы:**
  1. **Недостаток разнообразия**: лишь крупные популярные достопримечательности, смещены в сторону Европы и Азии.
  2. **Смещения (bias)**: ориентация модели под туристические фото, затруднения с малоизвестными или сельскими местами.
  3. **Пропуски**: отсутствуют мелкие объекты, улицы, здания менее известные.
  4. **Разрешение изображений**: модели CV чувствительны к низкому качеству — возможны ошибки при плохой освещенности или плохом разрешении.
- **Решения:**
  - Расширить базу точек путем добавления тысяч дополнительных достопримечательностей (например, из OpenStreetMap, Wikidata).
  - Аугментация изображений: добавить шум, поворот, изменение яркости, чтобы модель была более устойчива.
  - Нормализация координат: агрегация близких точек в кластеры для снижения шума.
  - Проверка валидных URL: все ссылки на изображения хранятся в формате `raw.githubusero` — надежны и доступны без авторизации.

4 Выбор модели

4.1 Описание модели GeoCLIP

- **GeoCLIP** основана на архитектуре CLIP (Contrastive Language-Image Pre-training), дообученной для задачи геолокации.
- Принимает на вход RGB-изображение, обрабатывает его через CNN-энкодер (ResNet-50 или ViT), получая вектор признаков.
- Вектор сравнивается с векторами GPS-координат (обычно через специальный геокодер/проекцию).
- Выдает топ-K ближайших координат с вероятностями (confidence scores).

4.2 Альтернативные модели

- **PlaNet (Google):** также предсказывает координаты по изображениям, но закрыт.
- **IM2GPS:** классический подход (2021) с использованием кластера визуальных признаков.
- **ResNet + kNN:** простой вариант: извлекаем эмбеддинги через ResNet, ищем ближайшие картинки в датасете с метками GPS.

- **Vision Transformer (ViT) + FAISS:** более современный энкодер, хранит индексы через FAISS.

4.3 Обоснование выбора GeoCLIP

- **Высокое качество предсказаний:** GeoCLIP обучалась на большом количестве изображений с метками GPS, демонстрирует высокую точность.
- **Простота интеграции:** доступна через PyPI, легко загружается и вызывает метод `model.predict(image, top_k)`.
- **Наличие вероятностных оценок (confidence scores).**
- **Альтернатива ResNet+kNN:** требует собрать большие датасеты изображений и строить базы кластера или индексы FAISS. GeoCLIP уже содержит проекцию в пространстве координат.

5 Системный дизайн

5.1 Пайплайн проекта

1. **Сбор данных / Загрузка модели:**
   - При запуске API сервис загружает веса GeoCLIP (пакет `geoclip`) и переводит модель в режим `eval()`.
   - PostgreSQL инициализируется скриптом `init.sql` (23 тестовые записи).
2. **Предобработка запроса:**
   - Клиент (Streamlit) отправляет изображение через multipart/form-data.
   - API сохраняет в `NamedTemporaryFile(.jpg)`, затем вызывает `predict_topk(model, path, top_k)`.
3. **Инференс модели:**
   - GeoCLIP читает файл, автоматически применяет нормализацию, ресайз, трансформация, пропускает через энкодер → получает топ-K координат с вероятностями.
   - В случае `/predict/coords` API возвращает JSON с массивом `[ (lat, lon, prob)...]`.
4. **Поиск ближайших точек (для `/search/nearby` и `/examples/nearby`):**
   - Берется одна координата (центр), передается в функцию `search_nearby(center, radius_km, db)`.
   - В `database.py` реализована функция Haversine для вычисления расстояния *в километрах* между двумя точками, фильтрация по радиусу, сортировка по возрастанию расстояния.
   - Возвращается список словарей {`id, name, lat, lon, url, distance_km`}.
5. **Формирование ответа и отображение:**

- API возвращает JSON: `{"center": {lat, lon}, "matches": [...]}`.
- Frontend: строит таблицу через `pandas.DataFrame`, отображает интерактивную карту Folium с маркерами.
- Tooltip для центра показывает загруженное изображение (base64), для остальных — картинку из поля `url`.

6. **Развертывание (Docker Compose):**
   - Сервис разбит на три контейнера: `db` (Postgres), `api` (FastAPI+Uvicorn), `frontend` (Streamlit).
   - Переменная окружения `DATABASE_URL` задается в `docker-compose.yml`.
   - Образы собираются из одного `Dockerfile`, в котором прописаны зависимости (FastAPI, SQLAlchemy, streamlit, folium и пр.).

5.2 Инструменты и технологии по этапам

Этап	**Инструмент / Технология**	**Описание**
Сбор данных / Загрузка модели	PyTorch, библиотека `geoclip`	Получение предобученных весов GeoCLIP
База данных (инициализация)	PostgreSQL, SQLAlchemy, `docker-entrypoint-initdb.d`	Сохранение и чтение метаданных (id, name, lat, lon, url)
API-сервис	FastAPI, Uvicorn, Pydantic	Обработка HTTP-запросов, валидация, генерация ответов в формате JSON
Логика поиска	Python, функция Haversine	Калькуляция расстояний и фильтрация по радиусу
Frontend	Streamlit, Folium, `streamlit-folium`, Pandas	Интерактивная визуализация результатов, карты, таблицы, предпросмотр изображений
Контейнеризация	Docker, Docker Compose	Разделение сервисов (DB, API, Frontend), упрощенное развертывание
Мониторинг и логирование	Возможное расширение: Prometheus, Grafana, ELK	(В MVP не реализовано, но запланировано для продакшен-версии)

5.3 Масштабируемость и производительность

- **Загрузка модели:** при старте API модель загружается в память (один экземпляр). Чтобы горизонтально масштабировать, можно запустить несколько реплик FastAPI за балансировщиком (NGINX, Traefik).

- **Инференс:** GeoCLIP inference может занимать 50–200 мс на CPU, 20–50 мс на GPU. При возрастающем числе запросов:
  - Горизонтальное масштабирование (несколько pod/контейнеров).
  - Использование очередей (RabbitMQ, Redis) для разгрузки синхронных запросов.
  - Кэширование популярных результатов (Redis).
- **Поиск в БД:**
  - Текущая реализация: линейный перебор всех записей (23). При росте базы до тысяч/миллионов точек потребуется индексирование:
    * Пространственные индексы PostGIS + `ST_DWithin` для быстрого поиска по радиусу.
    * Использовать `cube` или `earthdistance` в PostgreSQL.
    * Или хранить точки в ElasticSearch с гео-запросами.
- **Frontend:** Folium рендерит карту на стороне клиента, загружает HTML/JavaScript. При большом количестве маркеров (>1000) производительность падает. Решения:
  - Кластеризация маркеров (Leaflet MarkerCluster).
  - Lazy loading: загружать только область в видимой области карты.

6 Оценка качества

6.1 Метрики качества предсказаний

- **Mean Distance Error (MDE):** среднее расстояние (в км) между предсказанной и истинной координатой для тестовых изображений.
- **Precision@K (P@K):** доля случаев, когда одна из топ-K предсказанных координат попадает в радиус r км от истинной точки.
- **Recall@K (R@K):** для каждого изображения — насколько часто истинная метка входит в топ-K.
- **Coverage:** процент изображений, для которых модель вернула координаты (без ошибок)

6.2 Метрики качества поиска

- **Hit Rate:** доля случаев, когда в базе найдены объекты в заданном радиусе.
- **Average Distance to Nearest:** среднее расстояние от предсказанной точки до ближайшей найденной достопримечательности.
- **Response Time:** среднее время отклика (latency) API для эндпоинта `/search/nearby`.

6.3 Процедура оценки

1. **Сбор тестового набора:** набор фотографий с известными координатами (не из базы обучения).
2. **Проводим inference:** получаем топ-K координат, вычисляем MDE и P@K.
3. **Оценка поиска:** для каждого изображения ищем ближайшие объекты из тестовой БД, считаем Hit Rate при разных r (1 км, 5 км, 10 км).
4. **Нагрузочное тестирование:** с помощью `locust` или `k6` тестируем `/search/nearby` и `/predict/coords`, измеряем 95-й перцентиль latency.

7 Этические и социальные вопросы

7.1 Приватность и персональные данные

- **Фото пользователя:** пользователь загружает своё изображение. Возможные риски: непреднамеренное раскрытие личных данных (лица людей, номера автомобилей, знаковые места).
- **Хранение загруженных фото:** в текущей реализации фото не сохраняются на сервере, а обрабатываются во временном файле и немедленно удаляются после инференса.
- **Рекомендация:**
  - Информировать пользователя о политике конфиденциальности.
  - Обеспечить SSL/TLS соединение (HTTPS) для защиты данных в пути.
  - Не хранить загрузки дольше, чем нужно для инференса.

7.2 Смещение и дискриминация

- **Смещение модели:** GeoCLIP обучена преимущественно на туристических фото популярных мест. Риск: плохое качество предсказаний для регионов, мало представленных в обучающей выборке (сельские местности, неанглоязычные регионы).
- **Решения:**
  - Расширить тренировочные данные, добавить больше фото из разных частей мира.
  - Проводить мониторинг ошибок модели в разных регионах.

7.3 Этические риски использования

- **Неправомерное использование:** геолокация по фото может быть использована для отслеживания людей без их согласия.
- **Риск ошибочной информации:** предсказание координат с погрешностью может привести пользователя в неверное место.
- **Решения:**

– В интерфейсе указывать диапазон неточности (например, ± 10 км).
– Дать возможность пользователю вручную скорректировать координаты.
– Включить disclaimer: «результаты носят справочный характер и могут содержать погрешности».