Verification Dashboard System Design

Johannes Skivdal, 2024-06-24

Dashboard modules

1. Show the video
2. Keyframes
   - Extracted using Sohail's tool
   - Can look through multiple
3. Geolocation map
   - Google Maps JavaScript library, or MapBox (I have experience with both but prefer the latter)
4. Metadata
   - Extracted as normal exif
5. Transcription and translation
   - OpenAI Whisper + GPT translation
6. Object Detection
   - Using CLIP, Yolo or equivalent. Would be nice to produce semantic vectors that are searchable at a later date
7. OpenStreetMap tags-based geolocation
   - Can be extracted automatically (using GPT for example), then used to search OSM to find the precise geolocation

Many of these tasks can and should be ran in parallel:

- Run large tasks concurrently using a task processing queue like Celery (Python)
  - Some tasks might need a GPU
- Getting a low TTFP (Time To First Paint): using websockets. Open WS to start processing, respond with partial results as they're available. Alternatively, most business logic could be on the frontend, with it coordinating the different tasks it needs done.
- Videos and images are uploaded and stored in an S3 bucket. Also added to a database table for use in relations
- Job results are stored to a database, cached results from last time served if availiable
- User should be able to hop back and forth between their previous videos

Tech stack

Frontend: React/Vue/Svelte or equivalent. Highly interactive frontend

- Tanstack Query hooks generation from OpenAPI spec provides nice integration. Kubb is a good option for this

Backend: Preferably FastAPI because of OpenAPI spec generation and being ASGI by default (as opposed to Django)

- Needs a database adapter/ORM, might use PeeWee or Tortoise, or just native connector
- Auth is a little finnicky (FastAPI insists on being stateless). JWT in cookie is probably the best option

Database: something that runs as a networked service (meaning we avoid SQLite) because of multi-process access

- PostgreSQL is a good option because the PostGIS and pgvector extentions might be useful in the future

Task Queue: Celery looks good for Python. Needs a coordinating broker service, Redis can be used.

- There's also a native Postgres pub/sub service with stored procedures availiable for making a proper task queue

Running

We run a web server process and multiple worker processes, a worker on a GPU server would be useful.

- Asking NREC for resources? They've got good automation potential through Terraform that would allow us to scale up and down dynamically for GPU resources
- Simple server (without GPU) always on running webserver, database, and s3 (minio)