Summary

Code

https://github.com/ankushb92/Voltage-Park-Assessment

Services

- API Service (api-service)
- Redis (redis-service)
- Inference worker (inference-worker)
- Inference engine (mochi-service)
- Storage Service (minio-service)
- Web App (web-app-service)

API Service

Exposes APIs: Submit Job, Enhance Job, Get Job Status, List Jobs, Get Generated Output File

Enhance Job: The default result of submit job is optimized for faster results (by running with a low inference count). If the user likes the result, they can call the Enhance Job API for that job and then we give them a higher quality output of the same video (by generating with a high inference count while keeping the seed and the prompt the same).

Redis

Used as an in-memory queue and also keeps track of the status of jobs.

We have the following queues (using redis lists):

- PENDING_JOBS
- PROCESSING_JOBS

We also maintain Job details (using redis hash) as follows:

```
key: <job_id>
mapping: {
  "submitted_at": <timestamp>,
  "prompt": <str>,
  "enhanced": <bool>,
  "seed": <int>,
  "status": CREATED | PENDING | PROCESSING | COMPLETED | FAILED
  "started_processing_at": null | <timestamp>,
  "completed_processing_at": null | <timestamp>,
}
```

(It would be ideal to use a relational DB like Postgres for maintaining Job details, especially for optimally querying the List Jobs API. I went with this approach in the interest of time).

Inference worker

Picks up jobs from the Redis queue and runs inference on those jobs.

Using async IO, we spawn multiple async tasks (which we call ‘workers’) that pick up jobs from the PENDING_JOBS queue, only if the length of PROCESSING_JOBS queue is less than a threshold (set to 4 for now).

Inference engine

I used this docker image I found for the Mochi model:
https://replicate.com/genmoai/mochi-1?input=docker

Deployed 2 replicas of the engine with 4 GPUs each.

(I wanted to use the official mochi repo directly and build my own service using it. However, I went with this approach because I was running out of time)

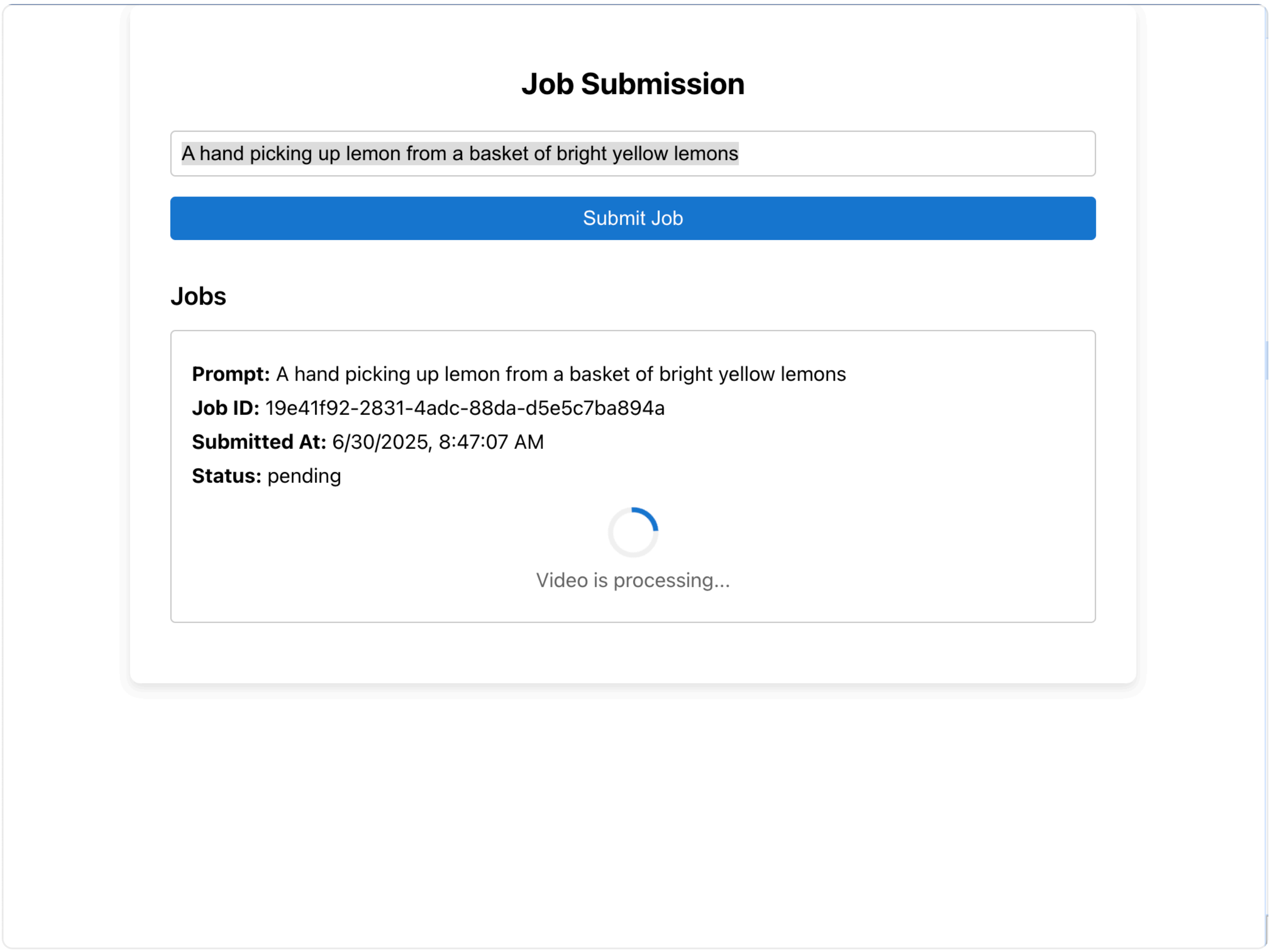
Storage Service

Used minio for object storage. Chose `minio` because it is S3 API compatible, which I’m familiar with.

Built a PV (persistent volume) of 200 Gi and attached a PVC to the minio deployment. The inference worker saves the model using the minio API and minio can generate links to the file that can be then served to the UI

Web App

A simple react app that allows user to enter a prompt and calls the `submit_job` API.
Once a job is submitted, it polls the `get_job_status` API every 5 seconds until it gets a `COMPLETED` status, and then displays the video. Any completed video can be enhanced with a higher inference count with the enhance button which calls the `enhance_job` API.


Job Submission

A hand picking up lemon from a basket of bright yellow lemons

Jobs

Prompt: A hand picking up lemon from a basket of bright yellow lemons
Job ID: 19e41f92-2831-4adc-88da-d5e5c7ba894a
Submitted At: 6/30/2025, 8:47:07 AM
Status: pending

Video is processing...

Deployment

Dockerfiles: `deploy/images/<service_name>/Dockerfile` (for each service)

k8s configuration: `deploy/k8s/<service_name>.yaml` has configuration for each service

Build script: `deploy/build_push.sh` : builds docker image and pushes to docker hub

Considerations/tradeoffs

- Performed benchmarking with a specific prompt (5 second video with inference count set to 100):

  8 GPUs: ~2.5 minutes to load and init model. ~2.5 minutes for inference
  4 GPUs: ~3 minutes to load and init model. ~4.75 minutes for inference
  2 GPUs: Took 4+ minutes to load and init model. ~9 minutes for inference
- Considered creating inference engine replicas with 2 GPU + 6 GPU combination, rather than 4 + 4. This would have led to greater GPU utilization during periods of low traffic.
- Should we reduce the number of inference steps during high load (when PROCESSING_JOBS queue is long)? This is not a good idea for `enhance_job` requests though.
- Content Safety

- GPU time-sharing: https://docs.nvidia.com/datacenter/cloud-native/gpu-operator/latest/gpu-sharing.html