Assignment A2: Data Processing Pipelines & IaC

Markson Chen (1009002598)
Kevin Liu (1008495237)
Barry Jiang (1008774097)
Yibin Cui (1008826512)

> *To recap: Our team aims to build an interactive virtual character system capable of real-time interactability combining speech, text, and motion. It involves designing a cohesive, low-latency, multimodal architecture that can (1) support modular SOTA **conversational models** end-to-end, (2) maintain continuous **visual grounding**, (3) convert visual input into reliable **navigation and motion plans**, (4) preserve and retrieve **memory** across sessions, and (5) orchestrate these asynchronous processes into a consistent, engaging, **real-time** agent experience.*

Part One: Aspirational Datasets

Among all the models our virtual character system uses, we find the biggest bottleneck is not motion generation but **visual grounding from continuously fed frames**. It is not difficult for the agent to understand a single image or even a single video using existing models that translate images or videos to text. However, it is much harder for it to **continuously update an internal visual understanding that is always up to date with the latest frame**, and there isn't a mature solution we can directly deploy.

To this end, we aim to adapt a pretrained video-language model (VLM) for **streaming captioning**: given recent video frames and previous captions, the model should generate new textual summaries of the visual input in a structured format.

If we have a "magic wand," the ideal dataset should be high-quality, consistent, and temporally structured, focusing on **incremental captions**. Starting from a more realistic dataset:

1. Descriptive Video Caption Dataset

Data schema: a set of videos, each paired with a **caption file**. The caption should aim for conciseness and maximally avoid repetition.

Example of captions:

- [00:00:00,000 → 00:00:05,000] A cow is at the dock, sitting, looking forward. The dock is made out of wood. The scene is a peaceful night scene.
- [00:00:05,000 → 00:00:07,000] The cow howls into the sky like a wolf.
- [00:00:07,000 → 00:00:12,000] The cow turns into a grey werewolf. The atmosphere turns grimy.

To finetune a VLM on this dataset, the input should be the current frames and the previous captions, and the output should be the caption for the current frames.

After finetuning, the model should get better at outputting structured, concise captions that capture incremental scene changes while avoiding repetitive scene descriptions.

2. Concise Captions with Human Preference

In addition to the data schema of the previous Descriptive Video Caption Dataset, this dataset will include multiple valid captions, each tagged with human-preference rankings that reflect redundancy/informativeness. Captions that are highly redundant should have a low preference.

We can use Reinforcement Learning from Human Feedback to finetune a VLM on this dataset, such that the conciseness of its generation aligns with human preference better.

3. High-Quality VR/Egocentric Dataset

Using the same data schema as Concise Captions with Human Preference, this dataset should contain large amounts of videos of egocentric VRChat gameplay. It should also contain edge cases of unstable camera motion and object interaction events.

This dataset is tailored for our AI character system that is to be deployed in VRChat. This training data will no longer be out-of-distribution with respect to the deployment scenarios.

Part Two: Reality Check

We find no datasets perfectly matching our use case (egocentric video with non-repetitive, descriptive captions). We are essentially proposing a novel task that outputs additive video descriptions for the goal of semantically complex and long-horizon video understanding. The datasets that do provide video captions are actually sports commentaries that are not semantically complex enough. Hence, we curated a custom dataset from existing public datasets listed below:

Item	Description	Commentary
Ego4D	A large-scale egocentric video dataset with first-person POV and temporal annotations.	First person POV is very aligned with our use case of getting visual input from VR.
Disney-Video Generation-Dataset	69 short (~6s) black and white cartoon videos.	Great for an initial experiment: Visually clean, motions are structured, and has no audio. Straightforward to generate streaming captions.
YouTube VRChat Gameplay Videos	Public VRChat gameplay recordings.	Perfect for realistic VR-style egocentric content aligned with our deployment environment.
Self-Recorded VR Sessions	Team-recorded VRChat or gameplay footage.	Most alignment with our final deployment environment. The scene changes are controlled and the supervision is structured.
Our Synthetic Dataset (curated from all above)	500 structured streaming captions generated using a large VLM (Gemini 3 Flash)	The core dataset for SFT and optional RL. Teaches in Molmo style with structured captioning as well as leveraging larger models that follows instructions well.

We follow a LIMA-style philosophy: small, high-quality, strong for aligning to a specific style while preserving model performance.

Part Three: Data-Processing Pipelines

3.1 Storage Layout

```
/raw/videos/
    video_001.mp4
    video_002.mp4
/processed/
    train.json
    rl.json
```

3.2 Dataset Schema

SFT Dataset:

```
{
      "video_id": video_id,
      "start_time": start_time,
      "fps": 4,
      "source_fps": float(source_fps),
      "window_duration": WINDOW_DURATION,
      "frames_window1": frames_w1_pil (image list),
      "frames_window2": frames_w2_pil (image list),
      "caption_window1": "A cartoon dog at a ship's wheel. A cartoon cow and
ducks on a dock.",
      "caption_window2_delta": "A duck is in a cage, screaming.",
      "conversations": ["user": ..., "assistant": ...],
}
```

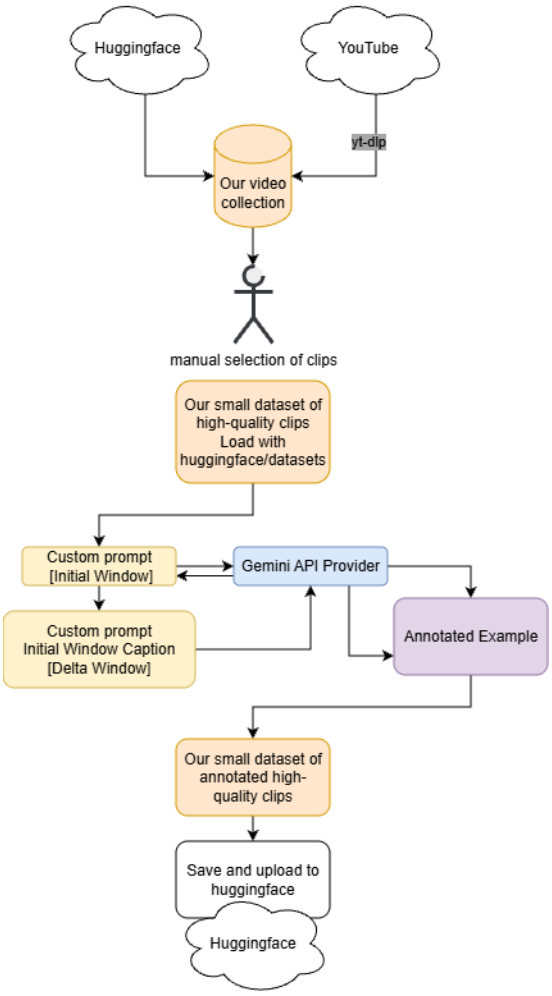
RL Dataset:

```
{
  "video_id": video_id,
  "start_time": start_time,
  "fps": 4,
  "source_fps": float(source_fps),
  "window_duration": WINDOW_DURATION,
  "frames_window1": frames_w1_pil (image list),
  "frames_window2": frames_w2_pil (image list),
  "objectives":["character enters room","object moved to table", "yellow
table"]
}
```

3.3 Pipeline Stages

1. **Select Videos** → `/raw/videos/`
2. **Frame Extraction** → TorchCodec or ffmpeg → frame tensors
3. **Initial Caption** → Gemini API (first window)
4. **Delta Caption** → slide window → Gemini API (with previous caption)
5. **JSON Construction** → append to `train.json`
6. **HuggingFace Dataset** → prepare for LoRA fine-tuning

3.4 Pipeline Diagram



3.5 Code (initial version)

See `synth_sft_gen.py`

Part Four: Infrastructure as Code Implementation

See our implementation at: https://github.com/UofT-CSC490-W2026/OpenNeuro/tree/main/infra

Since our project is a local desktop app that runs on the user's computer, we don't have as much infrastructure as a traditional SaaS platform. Therefore, our IaC stack is primarily for hosting cloud models that can be accessed through an API, which serves as an alternative option to running all models locally, in case our user doesn't have enough vram to run everything locally.

Currently, our IaC stack consists of the following:

1. Modal for model deployment. Currently, we deployed the Molmo2-4b model using vLLM.
2. Terraform for setting up AWS Lambda, which we use as an API gateway server that controls rate limit, billing, authentication & authorization, etc., for our cloud-hosted models.

Part Five: Disaster Recovery Demonstration

Watch the demo here!