Developing An AI-Powered Video Anonymization Pipeline

Bergo
Computer Science Department
Cal Poly Pomona
Pomona, USA
blbergo@cpp.edu

I. Introduction

Over the past two years, the Mobility Scooter Lab at Cal Poly Pomona has been collecting patient data across a variety of modalities while they are operating a mobility scooter. This data is primarily used to evaluate patient metrics that correspond to the WHO International Classification of Functioning, Disability and Health (ICF) and SNOMED frameworks [1].

Beginning in September of this year (2025), a joint effort began in collaboration with the University of California Riverside to explore how multi-modal LLMs can be used to provide oversight for traditional machine learning models. In order to facilitate the safe transfer of patient data, as well as to prepare it for eventual publication as part of our NSF grant, I was tasked with building a video anonymization pipeline.

Using a combination of machine learning and high-performance compute technologies, I was able to successfully build and deploy this pipeline to the Delta HPC system [2]. What follows is detailed description of my approach to solving this problem, as well as the challenges I encountered while implementing my solution. My work can be found here, and please request access from tingtingchen@cpp.edu if required as this repo contains some patient data.

II. Literature Survey

The most significant piece of literature relevant to my project is the original paper depicting the architecture of the YOLO series of models. The advantage of these object-detection models in comparison to previous models is that they compute bounding boxes and classes in one pass (hence "You Only Look Once") [3]. While the pipeline is offline in nature, HPC is not cheap, and the YOLO models typically have higher performance for a metric called intersection over union (IoU), which determines when the model decides if two detected objects are overlapping or are a single object.

Other forms of literature include the Nvidia PyNvVideoCodec and DALI documentation, which were instrumental in guiding me through understanding the process of hardware-based video encoding and decoding [4] [5]. The Kornia Geometric and Computer Vision Library documentation library also enabled me to better understand how color spaces are implemented on the GPU [6]. Lastly, the Dask-Jobqueue library enabled me to distribute the pipeline across an arbitrary number of Nvidia A40 GPUs on Delta, which are optimized for encoding-heavy workloads [7].

III. Project Goals

As previously mentioned, the objective of this pipeline is to remove any personally identifiable information of the patients from the videos of their driving sessions. For our purposes, we identified patient faces and car license plates as relevant PII present in the video data, and removed all audio tracks. We initially prioritized the mean-accuracy precision score across confidence intervals 5-95 (mAP5-95B), and later shifted to recall in order to minimize the number of face exposures.

A secondary goal of the pipeline is to ensure that it is both hardware and time efficient in order to fulfill data requests from the UCR team in a reasonable amount of time. While preempted jobs are generally cheaper to run on Delta, an end-of-day data request may necessitate interactive jobs, which can cost up to 12x more depending on the hardware [8]. For this reason, it was crucial that the pipeline be developed in a modular way, such that it can be scaled up or down to meet the time or cost efficiency requirements.

IV. Methodology

Over the course of the semester, I developed two iterations of this pipeline, with each being used to process roughly 40 videos respectively. The general flow for both pipelines is depicted in Fig. 1.

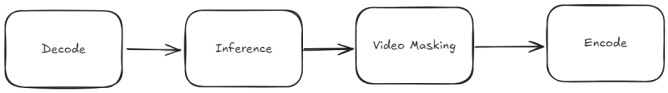

Fig. 1: High-level Pipeline Architecture

In short, video frames are decoded on the GPU, resized to the optimal input size for a fine-tuned YOLO model, whose outputs are then used to create blurring masks which are applied to the frames before they are encoded into a new file.

A. Iteration 1

The first iteration was created in just over the span of a month, initially utilizing GPU decoding, resizing, and inference, with masking and encoding taking place on the CPU. For a first iteration, this performed decently, averaging 20

frames per second for end-to-end processing. With our dataset containing roughly 6.5 million frames, this would take just over 90 hours to process with a single A100. For spontaneous requests, this would not suffice, especially given that our data collection is only half-way complete.

B. Iteration 2

The second iteration focused entirely on hardware (GPU) based improvements. I swapped PyNvVideoCodec for the Nvidia Data Loading Library (DALI), which allowed frames to be batched, normalized, and resized as a stream, whereas the previous pipeline did these operations sequentially per batch before performing inference.

I also migrated from the ffmpeg-python library to PyNvVideoCodec's implementation of hardware-accelerated video encoding, which required me to build both ffmpeg and the h264 codec from source on Delta. For stability, edited videos were first encoded to h264 format with a NV12 colorspace before being re-encoded into mp4 files. While this introduced a slight amount of overhead, it ensured that videos with corrupted frame indexes could still be processed by the pipeline. With these additions, frames were completely processed on the GPU with zero copy tensors, resulting in a batch processing time of 90 FPS.

V. Results and Evaluations

While most of the attention thus far has been paid to the pipeline, it is important that we recognize the metrics of the models used to produce the bounding boxes and class identifiers that are eventually blurred. I tested a variety of YOLO-compatible models, including YOLO8, YOLO11, YOLO12 and the transformer-based RT-DETLR. Each of these models were fine-tuned with a set of 400 examples collected from our dataset and annotated with Label Studio, with a ratio of 2:1 faces per license plate [9].

Each of these models scored in the high 30s for mAP5-95, so I shifted to fine-tuning models which were pretrained on a dataset that contained either faces or license plates. The table below shows these results on a test set of 10% of the entire dataset.

Model	Test mAP5-95 (B)
yolov8l-oiv7	0.73
yolov12m-face	0.81

TABLE I: Fine-tuning results

Note: yolov12m-face was only fine-tuned for face detection, and is used in iteration 2 alongside yolov8l-oiv7 for ensembling low-confidence face detections. In practice, this results in incredibly rare face exposures, however when they do occur, they tend to be for 30-60 ms, which is unacceptable for publishing.

VI. Conclusions and Future Work

The results of the pipeline are very promising, however more effort is required to increase throughput, and more importantly, increase the accuracy of the base model. License plates in particular lag behind face mAP in test evaluations, hovering around 0.4, which is likely due to the compression that is applied to each video before it is uploaded.

Unfortunately, optimizing the data ingestion process is outside of the scope of this project, although it could be a new avenue for increased accuracy for plates. Throughout my fine-tuning runs, models performed consistently better when the input was upscaled by 50% or higher, which may be a sign that a higher resolution is required to achieve improved results.

The last area I am currently exploring is related to how each of the data splits is generated. I initially increased the accuracy of fine tuning by gathering examples of patient face exposures, however I realized that random shuffling for train/test/val splits would not guarantee an equal distribution of these outlier exposures.

To address this, I implemented a Vector-Quantization Variational Autoencoder (VQ-VAE) [10]. After training this model on the unlabeled dataset, I was able to create clusters based on the latent codes outputted by the encoder and sample from each cluster to form each split. While this produced better results, when I attempted to validate a model trained on the original dataset with this new dataset, it outperformed the new model by nearly 20%. This signified to me that this approach lacked robustness, and was not a suitable avenue to continue upon.

The shortcomings of the VQ-VAE inspired me to pivot my approach to collect a uniform number of samples from each video, label them with my best existing model, and assign each video to a training split. While this process may not be achievable by the end of the semester, it does lay the foundation for the next step in the development of the pipeline alongside an increase focused on improving the recall of the model.

References

[1] SNOMED CT https://www.nlm.nih.gov/healthit/snomedct/index.html
[2] Delta https://www.ncsa.illinois.edu/resources-and-services/compute-resources/delta/
[3] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.
[4] PyNvVideoCodec Documentation https://developer.nvidia.com/pynvvideocodec
[5] Nvidia DALI Documentation https://docs.nvidia.com/deeplearning/dali/user-guide/docs/index.html
[6] Kornia GitHub https://github.com/kornia/kornia
[7] Dask-Jobqueue https://jobqueue.dask.org/en/latest/
[8] Running Jobs https://docs.ncsa.illinois.edu/systems/delta/en/latest/user_guide/running_j partitions-queues
[9] Label Studio https://labelstud.io/
[10] Van Den Oord, Aaron, and Oriol Vinyals. "Neural discrete representation learning." Advances in neural information processing systems 30 (2017).