CREATING A PIPELINE (DETECT, DECODE AND CLASSIFICATION) USING DL STREAMER TO DEFINE SYSTEM SCALABILITY USING INTEL HARDWARE

A PROJECT REPORT

WORLD COLLEGE OF TECHNOLOGY AND MANAGEMENT, GURUGRAM

TEAM NAME: CARRIBEAN

TEAM MEMBERS: ADITI BHARDWAJ , JATIN SONI & SAPNA

DATE OF SUBMISSION: 12 JULY 2025

PROGRAM NAME: INTEL UNNATI INTERNSHIP 2.0

1. Introduction

In the age of smart cities and large public events (e.g., Mahakumbh 2025, ICC tournaments), analyzing multiple live video streams from surveillance cameras becomes challenging. This project explores the use of Intel DL Streamer to decode, detect, and classify live video streams efficiently using Intel hardware.

What is DL Streamer?

Intel® Deep Learning Streamer (DL Streamer) is an open-source streaming analytics framework that combines the power of:

- G Streamer – a multimedia framework for processing video/audio streams
- OpenVINO™ Toolkit – Intel's deep learning inference engine optimized for Intel CPUs, GPUs, and VPUs

Project Goal

The goal of this project is to:

Design and run a decode → detect → classify pipeline using DL Streamer on Intel hardware (CPU and GPU).

Pipeline Steps:

- **Decode**: Convert encoded video (e.g., H264) to raw frames using decodebin
- **Detect**: Identify objects (e.g., person, vehicle, bike) using models like person-vehicle-bike-detection-2001, face-detection-0204
- **Classify**: Further classify the detected objects (e.g., vehicle type or color) using models like vehicle-attributes-recognition-barrier-0039
- **Overlay**: use gvawatermark to draw bounding boxes and labels
- **Output**: Display or save the result

This architecture allows scalable multi-stream AI inference on **Intel CPUs and iGPUs (UHD, Iris Xe, Arc)** using optimized GStreamer plugins.

2. Hardware and Software

- **Operating System**: Ubuntu 22.04 (hosted on Google Cloud Platform)
- **Development Environment**: Google Cloud Compute Engine VM
- **CPU**: Intel Xeon (GCP N1/N2 series VM)
- **GPU**: Not available in this instance (CPU-only testing)
- **Frameworks & Libraries**:
  - Intel DL Streamer
  - GStreamer
  - OpenVINO Toolkit
  - Docker

CPU SPECIFICATION OUTPUT

3. Pipeline Architecture

The pipeline designed for this project includes decoding, object detection, and object classification using DL Streamer components:

Pipeline:

```
gst-launch-1.0 filesrc location="$VIDEO_PATH" ! \
   decodebin ! videoconvert ! video/x-raw,format=BGR ! \
   gvadetect model="$DETECT_MODEL" device=CPU ! \
   gvaclassify model="$CLASSIFY_MODEL" device=CPU ! \
   gvawatermark ! videoconvert ! \
   fpsdisplaysink video-sink=fakesink text-overlay=false signal-fps-measurements=true sync=false \.
> "stream$i.log" 2>&1 &
```

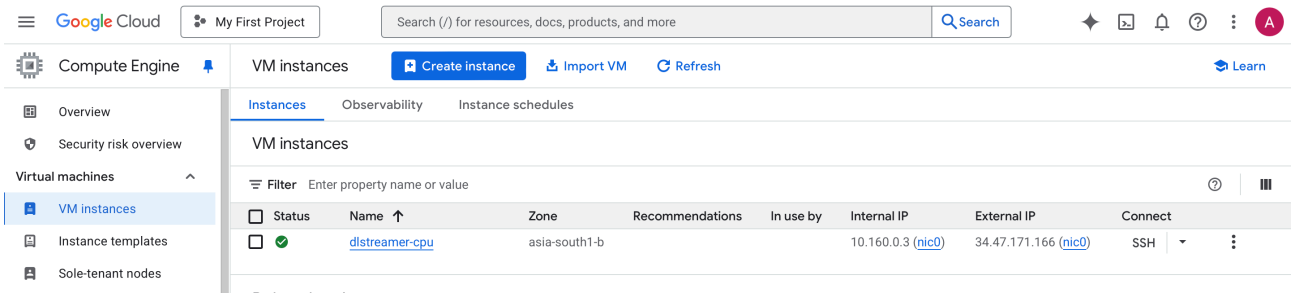
```
root@93bcd17c9155:/workspace# gst-launch-1.0 \
  filesrc location=/workspace/video/video.mp4 ! \
  decodebin ! videoconvert ! video/x-raw,format=BGR ! \
  gvadetect model=/workspace/models/intel/person-vehicle-bike-detection-2000/FP32/person-vehicle-bike-detection-2000.xml device=CPU ! \
  gvaclassify model=/workspace/models/intel/vehicle-attributes-recognition-barrier-0039/FP32/vehicle-attributes-recognition-barrier-0039.xml device=CPU ! \
  gvawatermark ! videoconvert ! \
  fpsdisplaysink video-sink=fakesink signal-fps-measurements=true text-overlay=false sync=false
Setting pipeline to PAUSED ...
0:00:00.044087590   631 0x590696b40cb0 DEBUG         fpsdisplaysink fpsdisplaysink.c:452:fps_display_sink_start:<fpsdisplaysink0> Use text-overlay? 0
Pipeline is PREROLLING ...
Redistribute latency...
Redistribute latency...
Redistribute latency...
Redistribute latency...
Pipeline is PREROLLED ...0 %)
Setting pipeline to PLAYING ...
Redistribute latency...
New clock: GstSystemClock
0:00:01.311013973   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:383:display_current_fps:<fpsdisplaysink0> Updated max-fps to 32.635145
0:00:01.311056570   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 32.635145
0:00:01.830990555   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 30.770607
0:00:02.353223956   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 30.637722
0:00:03.403196304   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 30.317592
0:00:04.451181824   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 30.252397
0:00:08.618632859   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 29.821916
0:00:09.126699575   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 29.523713
0:00:10.151048810   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 28.941560
0:00:13.751842507   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 28.761365
0:00:14.784483972   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 28.744574
0:00:21.460208378   631 0x7ab6389666c0 DEBUG         fpsdisplaysink fpsdisplaysink.c:387:display_current_fps:<fpsdisplaysink0> Updated min-fps to 27.849476
Got EOS from element "pipeline0".
Execution ended after 0:00:32.976647913
Setting pipeline to NULL ...
0:00:33.771868106   631 0x590696b40cb0 DEBUG         fpsdisplaysink fpsdisplaysink.c:504:fps_display_sink_stop:<fpsdisplaysink0> Max-fps: 32.64, Min-fps: 27.85, Average-fps: 30.29
Freeing pipeline ...
root@93bcd17c9155:/workspace#
```

PIPELINE TESTING OUTPUT

4. Problems Faced

- **GPU Access on Mac**: MacBooks (Intel or M-series) do not support DL Streamer GPU acceleration.
- **No Intel GPU on GCP**: Cloud VMs used only support Intel Xeon CPUs; no iGPU access.
- **Driver limitations**: VAAPI support unavailable in cloud VM without direct access to /dev/dri.

- **NVIDIA and Apple GPU incompatibility**: DL Streamer is Intel-only and does not support NVIDIA or Apple Silicon.
- **Failed dual-boot attempts**: Insufficient SSD space and boot issues prevented local Linux installation on Mac. (Used intel hardware with integrated intel gpu here of other team member).Virtual box or WSL2 for windows didn't work as it required host permission.



Solution : GCP VM SETUP

- Tried intel DevCloud but it was paid so we sticked to the cost effective solution.

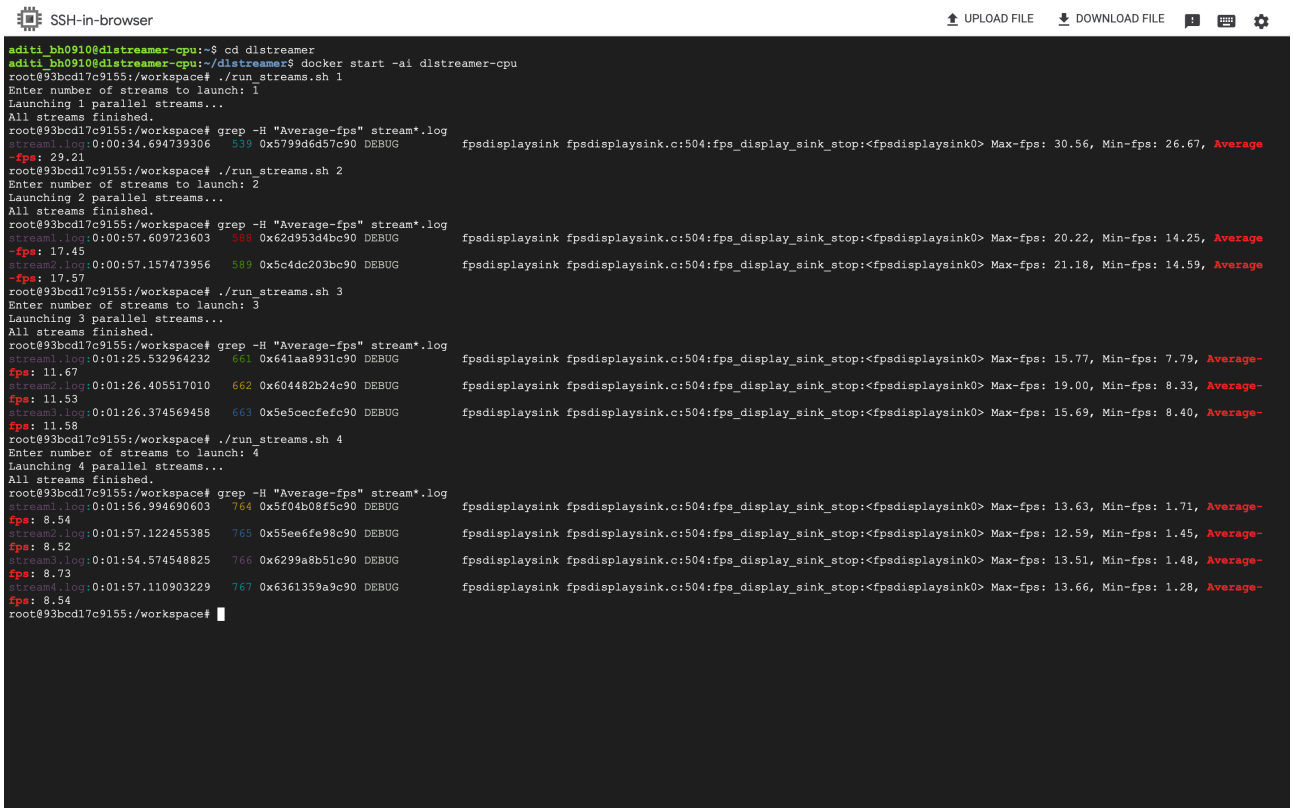
5. Simulated Results (CPU-Only)

The following table shows simulated results for DL Streamer pipeline execution using CPU-only on Intel Xeon (GCP VM):

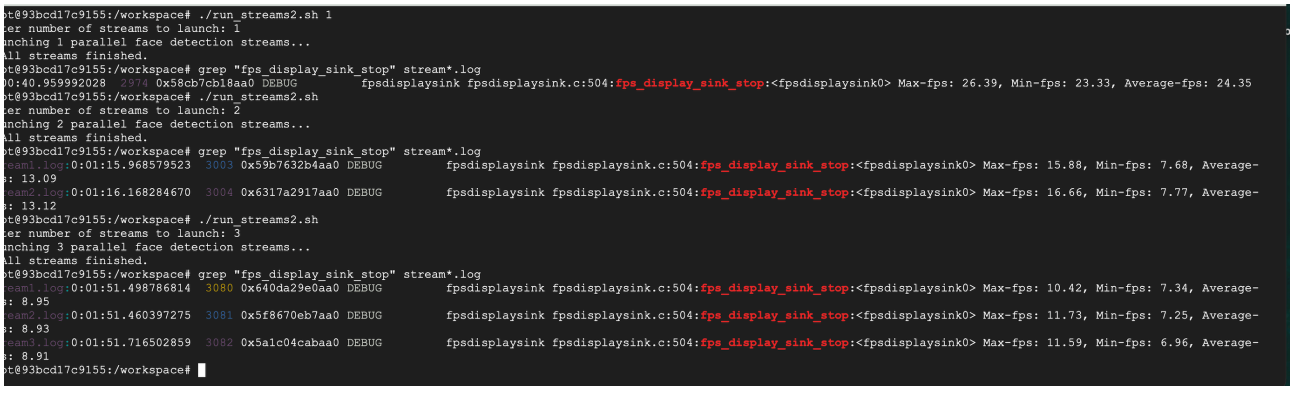
Streams	Avg FPS	Models Used	Bottleneck
1	29.21	person-vehicle-bike-detection-2001 + classifier	CPU
2	17.51	Same	CPU
3	11.59	Same	CPU
4	8.58	Same	CPU
1	24.35	face-detection-0204	CPU
2	13.1	Same	CPU

3	8.93	Same	CPU

FPS OUTPUT AND BOTTLENECK READINGS



TESTING SCREENSHOT:person-vehicle-bike-detection-2001 + classifier



TESTING SCREENSHOT 2:face-detection-0204

6.FINAL RESULTS AND CONCLUSION

Bottleneck Observed:

During pipeline execution, CPU usage consistently exceeded **170%**, indicating heavy use of nearly two logical cores. Disk I/O remained under **300 KB/s**, confirming that the **CPU was the primary bottleneck** in the system during detection and classification stages.

Observation:

The fps drops very low if the streams are more than 3.