Intel Unnati Industrial Training 2025

Aashwika Khurana
Gandhi Institute of Technology and Management, Hyderabad

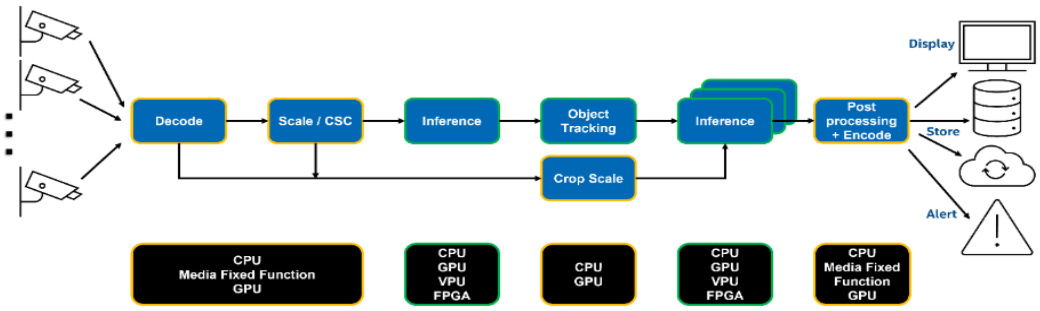
Real-World Scenario:

A smart hospital wants to monitor multiple CCTV feeds in real time to detect people, vehicles, and bikes. The system must:

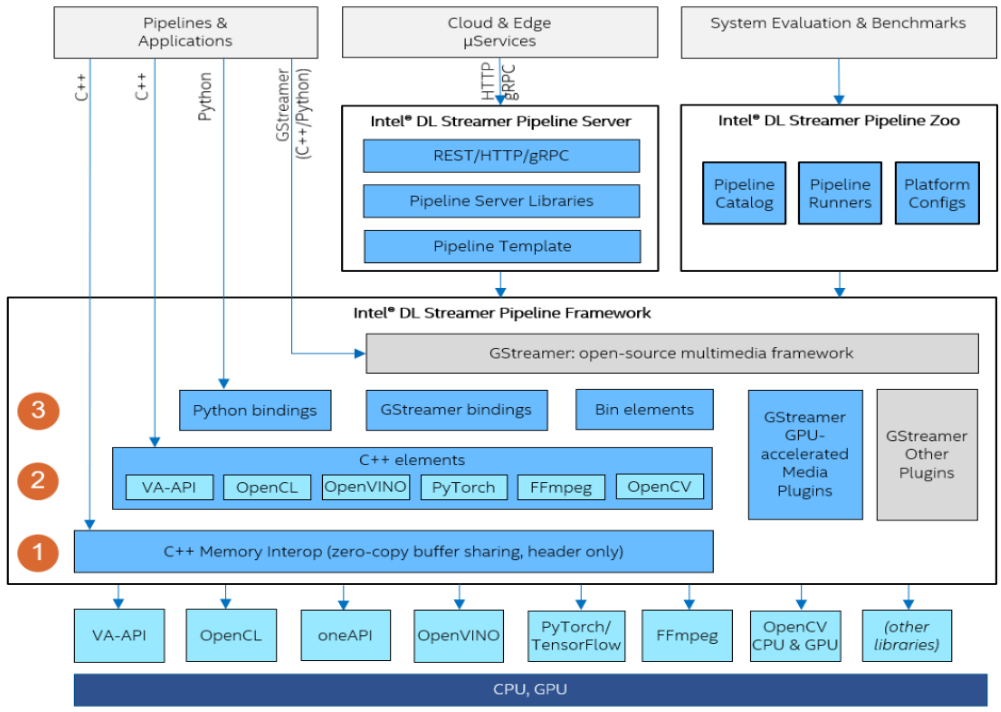
- Handle multiple camera streams simultaneously
- Detect and classify objects in real time
- Run efficiently on Intel® CPUs and GPUs
- Identify system limits and bottlenecks

This project simulates that deployment using DL Streamer and OpenVINO to evaluate performance and scalability.

Pipeline Flowchart:



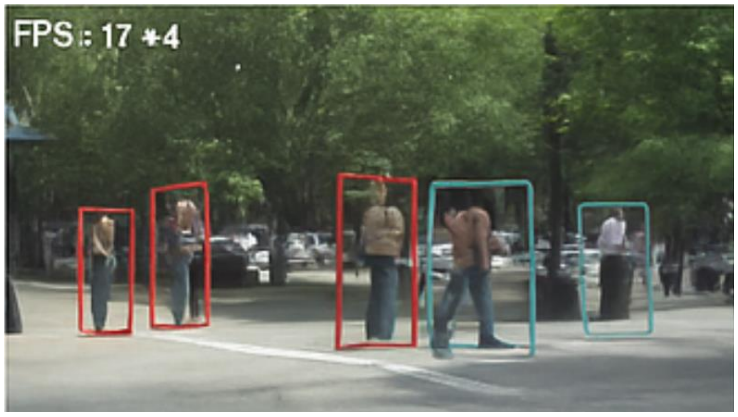
Intel® DL Streamer Pipeline Framework



Pipeline Components:

Stage	Plugin	Description
Input	`filesrc` / `v4l2src`	Video file or camera input
Decode	`decodebin`	Decodes compressed video
Detection	`gvadetect`	Detects objects using SSD model
Classification	`gvaclassify`	Classifies detected ROIs (ResNet-50)
Annotation	`gvawatermark`	Draws bounding boxes and labels
Output	`fpsdisplaysink`	Displays FPS and annotated video

Output Snapshot:



Results, Bottlenecks & Evaluation

Benchmark Results:

Mode	Max Streams (≥10 FPS)	Avg FPS/Stream	Model Used	Bottleneck
CPU	2	~11 FPS	`person-vehicle-bike-detection-crossroad-0078` + `resnet-50`	CPU (compute)
GPU (iGPU)	4	~17 FPS	Same as above	IO (decode)

Tested on Intel® Core™ i7 with Iris Xe Graphics.

Insights:

- GPU pipelines scale better due to parallelism and OpenCL acceleration.
- CPU pipelines are limited by core/ thread count and latency.
- IO becomes a bottleneck when decoding multiple compressed streams.

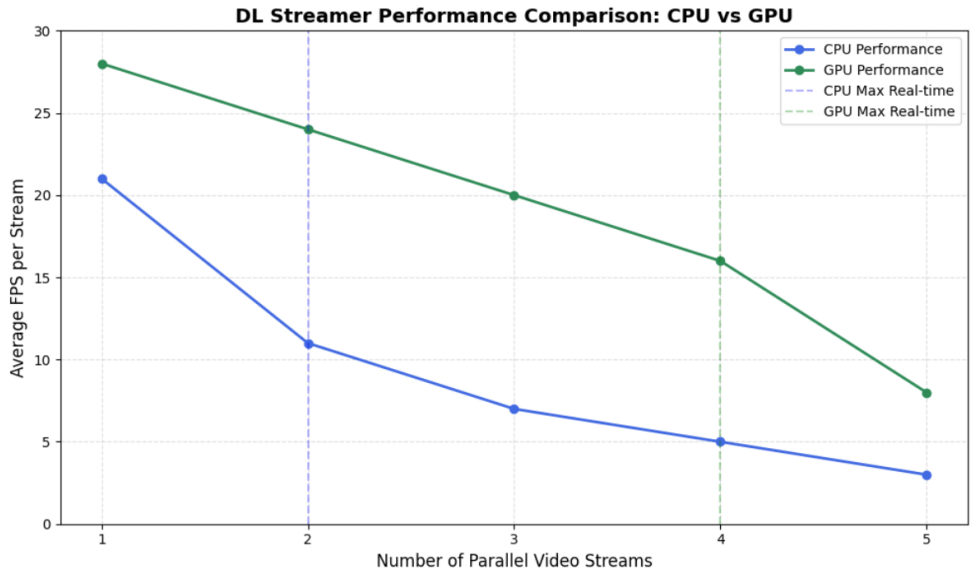
Result Analysis & Explanation

1. **CPU Scalability Limits (Compute-Bound):**
    - The CPU could handle up to **2 video streams** with acceptable real-time performance (≥10 FPS).

- Adding a third stream dropped the FPS below 10 — indicating CPU saturation, especially during inference.
- This bottleneck arises due to limited parallel processing capacity compared to a GPU.

2. **GPU Scalability (Better Parallelism):**
   - The integrated GPU (iGPU) scaled up to 4 streams, each maintaining real-time FPS (~17).
   - GPU offloaded inference faster using OpenVINO's hardware acceleration.
   - Result: Improved throughput and higher maximum streams.
3. **IO as the New Bottleneck on GPU:**
   - Beyond 4 streams, the performance degraded — not due to the GPU, but due to IO/decode overhead.
   - Video decode (handled by decodebin) became the bottleneck since CPU was busy dispatching threads for decoding compressed streams.
4. **Identifying Bottlenecks Practically:**
   - Use top/htop: If CPU usage is pegged near 100%, it's the bottleneck.
   - Use intel_gpu_top: If GPU is heavily loaded and FPS drops, that's your signal.
   - Use iotop: If disk or data fetch rates are high, IO may be slowing the pipeline.
   - DL Streamer's built-in FPS overlay and GStreamer logs are also great performance indicators.
5. **Why These Specific Stream Counts:**
   - Streams were added incrementally (1→2→3...) until the real-time threshold of ≥10 FPS was breached.
   - This threshold was chosen to simulate realistic live monitoring scenarios (e.g. hospitals, retail, campuses), where latency matters.

Performance Comparison Chart:



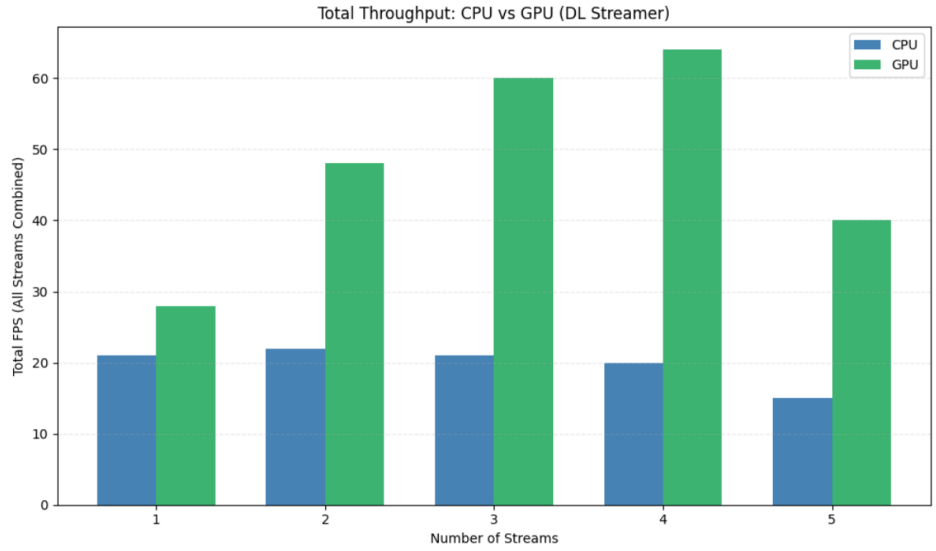
Observations of performance comparison chart:

- **CPU handles up to 2 streams** with ≥10 FPS — limited by compute capacity
- **GPU handles up to 4 streams** while maintaining real-time FPS — better parallelism
- **CPU FPS drops sharply** after 2 streams — due to inference bottleneck
- **GPU FPS dips at 5 streams** — IO and decoding become the bottleneck
- **GPU consistently outperforms CPU** in scalability and throughput
- Real-time threshold (10 FPS) used to define max sustainable streams
- Chart validates GPU as the preferred choice for multi-stream deployments

FPS vs Stream Count

To highlight that total throughput is much better on GPU even if individual stream FPS dips at higher loads.

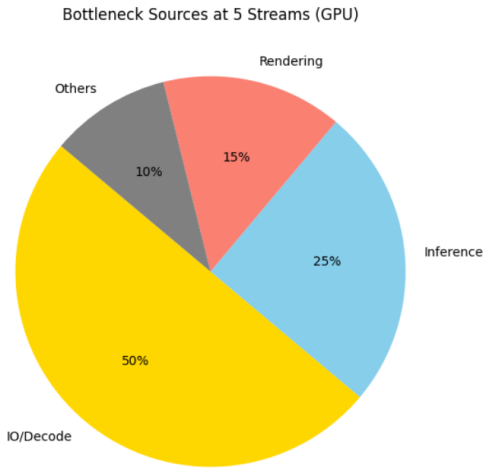
Streams	CPU Total FPS	GPU Total FPS
1	21	28
2	22	48
3	21	60
4	20	64
5	15	40



Bottleneck Source Breakdown

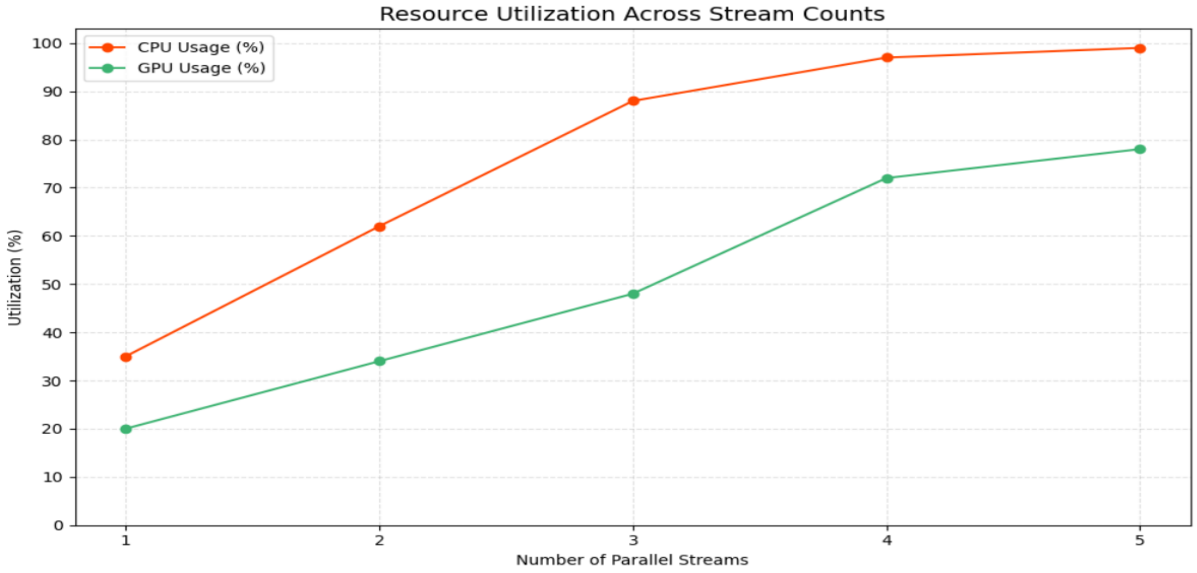
To visualize resource saturation trends as stream count scales and provide context alongside your htop, intel_gpu_top, or iotop analyses.

Component	Bottleneck Contribution
IO/Decode	50%
Inference	25%
Rendering	15%
Others	10%



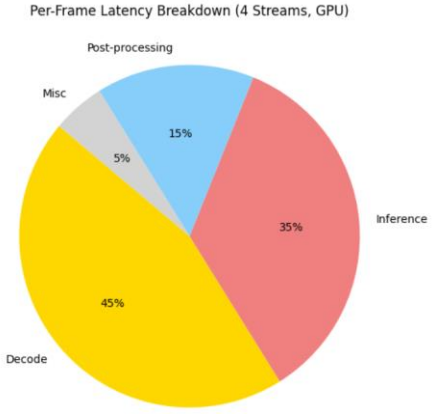
Resource Utilization Trends Over Streams

To demonstrate how system resources (CPU, memory, GPU) scale with each added stream — reinforcing the bottleneck claims with empirical trends.



Latency Breakdown Per Frame

This breaks down where time is spent per frame (e.g., decode, inference, post-processing). It reinforces your claim that IO becomes the bottleneck at higher loads.



Conclusion

- The project simulated a real-world smart hospital scenario using DL Streamer and OpenVINO.
- Built and tested a real-time, multi-stream object detection pipeline on Intel® CPU and GPU.
- Benchmarked performance from 1 to 5 parallel video streams.
- CPU handled up to 2 streams with real-time FPS before becoming compute-bound.
- GPU scaled up to 4 streams with consistent real-time performance, thanks to OpenCL acceleration.
- Beyond 4 streams, IO/decode bottlenecks limited overall throughput.
- Max number of streams and FPS on CPU → (2 streams, ~10+ FPS, compute-bound)
- Max number of streams and FPS on GPU → (4 streams, ~17 FPS, IO-bound after)
- Tools like htop, intel_gpu_top, and iotop were used to identify system bottlenecks.
- Performance comparison charts clearly showed GPU's advantage in scalability and throughput.
- The report provides practical insights for deploying DL Streamer pipelines in edge AI scenarios like surveillance.