Intel® Unnati
Data-Centric Labs in Emerging Technologies

EdGate TECHNOLOGIES

PROJECT REPORT ON

AI Video Analytics Pipeline using DL Streamer on Intel Hardware

Submitted By

KANNEBOINA RAJASHEKAR (237Y1A05P5)

C.P.MISHRA(237Y1A0563)

COMPUTER SCIENCE AND ENGINEERING

Marri Laxman Reddy Institute of Technology and Management

Under the Guidance of

Kattaboyina Anusha, Assistant Professor

Submitted to

Intel® Unnati Program

Supported by

Edugate Technologies

In partnership with

Marri Laxman Reddy Institute of Technology and Management (MLRITM)

EMPOWERING THROUGH INNOVATIONS
MLRS

Date of Submission: July 2025

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

With the growing implementation of Smart City initiatives and the rising security demands at large-scale public events such as the MahaKumbh Mela and international sports tournaments, AI-powered camera systems are becoming critical. These systems are designed to analyze live video feeds in real-time to detect, classify, and track objects such as people, vehicles, or suspicious activities.

In scenarios involving hundreds or thousands of surveillance cameras, manual monitoring is infeasible due to the enormous volume of data and the need for immediate response. Artificial Intelligence (AI) addresses this challenge by automating the video analytics pipeline, providing accurate and timely insights to authorities and system operators.

This project aims to develop a deep learning pipeline using Intel's DL Streamer and OpenVINO™ Toolkit that performs the following tasks:

- Decoding video streams,
- Object Detection (e.g., person, vehicle),
- Object Classification (e.g., type of object or behavior).

The pipeline is deployed and tested on Intel hardware (CPU and GPU) to evaluate system performance. The primary goal is to assess the scalability of the system by:

- Measuring the maximum number of supported camera streams,
- Recording the frames per second (FPS) under various conditions,
- Identifying bottlenecks in the pipeline (CPU, GPU, or I/O).

By focusing on real-world use cases like the MahaKumbh Mela, this project demonstrates how AI and optimized deep learning pipelines can enhance public safety, support crowd management, and drive smart city applications efficiently at scale.

Figure 1.1 A crowded MahaKumbh Mela scene with thousands of people.

CHAPTER 2

LITERATURE SURVE

The increasing deployment of AI-based video surveillance systems in public and smart city environments has driven the need for scalable and efficient video analytics solutions. Events such as the MahaKumbh Mela 2025 and the ICC Men's T20 World Cup 2024 have demonstrated the critical role of AI-powered camera systems for crowd monitoring, security, and operational analytics.

Traditional surveillance systems are not scalable due to the extensive human effort required to monitor thousands of real-time video streams. Recent developments in machine learning (ML) and deep learning (DL) — especially in object detection and classification models like YOLO (You Only Look Once), SSD (Single Shot Detector), and MobileNet — have enabled accurate and real-time analysis of streaming video data. These models can detect persons, vehicles, and other objects efficiently, making them well-suited for smart surveillance.

However, performing inference at scale requires significant compute resources. This has led to the rise of edge computing, where AI tasks are executed close to the video source using high-performance edge devices, such as Intel CPUs, GPUs, and VPUs. According to Intel's documentation and technical use cases, the OpenVINO™ Toolkit and DL Streamer provide highly optimized support for deploying deep learning pipelines on Intel hardware. These tools support essential video analytics tasks such as decoding, detection, and classification, integrated into modular GStreamer pipelines.

Real-World Applications --

MahaKumbh 2025: AI-powered facial recognition and crowd density analysis are helping law enforcement ensure public safety through real-time.

-Surveillance (LiveMint, 2024).ICC Men's T20 World Cup 2024: AI-driven video production systems assist in object tracking, event classification, and in-game analytics, enhancing both broadcast quality and operational insight (ICC Media, 2024). Studies from organizations like IEEE Smart Cities have consistently emphasized the importance of distributed video analytics systems that run at the edge. Intel's **DL**

Streamer has emerged as a reliable and scalable solution to support
Multiple parallel camera streams,
Real-time frame rates (FPS),
Load balancing across CPU, GPU, and I/O resources
In summary, integrating DL Streamer with OpenVINO on Intel hardware presents a practical and scalable approach to building real-time AI surveillance systems. This has been validated in both experimental and large-scale real-world deployments, confirming its relevance and applicability in modern smart cities and public safety projects

CHAPTER 3

SYSTEM REQUIREMENT SPECIFICATION

3.1 Tools and Technologies

Tool / Technology	Description
DL Streamer	Open-source streaming analytics framework optimized for Intel hardware; used for decoding, inference, and rendering in video pipelines.
OpenVINO™ Toolkit	Intel toolkit for optimizing and deploying deep learning inference on edge devices (CPU, GPU, VPU).
OpenCV	Library for real-time computer vision tasks; used for video capture and output display.
Python	Programming language used to implement the pipeline and control components.
Intel CPU / GPU	Hardware platforms used for running and benchmarking the pipeline's performance.

3.2 Hardware Requirements

Component	Specification
Processor (CPU)	Intel® Core™ i7 / i9 or Xeon with AVX2/AVX-512 support
Graphics (GPU)	Intel® Integrated GPU (Iris Xe) or Intel® Arc / discrete GPU
Memory (RAM)	Minimum 16 GB (32 GB recommended)
Storage	SSD with 100 GB free space for video files and logs

Component	Specification
Camera Input	Local video files or IP camera streams (simulated for testing)

3.3 Software Requirements

Software	Version / Details
Ubuntu OS	20.04 LTS or higher
Python	3.8 or higher
OpenVINO Toolkit	2022.3 or later
DL Streamer	Latest release from GitHub
OpenCV	4.x version with Python bindings
GStreamer Plugins	Required for DL Streamer pipeline execution

CHAPTER 4

SYSTEM DESIGN

4.1 Pipeline Architecture

The goal of this system design is to enable efficient, real-time video analytics across multiple camera streams using a deep learning pipeline built with DL Streamer and OpenVINO on Intel hardware (CPU/GPU). The system is designed to process high-resolution video inputs in parallel, performing object detection and classification while maintaining real-time frame rates.

The pipeline consists of the following key components:

4.2 Pipeline Stages

1. **Video Decode**
   Video streams are read using **OpenCV** or DL Streamer GStreamer elements, which decode video files or live camera inputs frame by frame.
2. **Preprocessing**
   Each frame undergoes preprocessing to prepare it for inference:
   - Resizing to the input dimensions required by the model (e.g., 300x300 or 416x416),
   - Normalization (scaling pixel values),
   - Format conversion (e.g., BGR to RGB, NHWC to NCHW if required by the model).
3. **Inference (Object Detection)**
   The preprocessed frames are passed to an **OpenVINO-optimized object detection model**, such as:
   - person-detection-retail-0013
   - ssd_mobilenet_v2_coco
   - yolov5s (OpenVINO-IR converted)
     These models return bounding boxes, class labels, and confidence scores.

4. **Postprocessing**

   The model output is parsed to extract:

   - Detected objects with confidence above a set threshold,
   - Coordinates for drawing bounding boxes,
   - Class labels (e.g., person, vehicle).

Bounding boxes and labels are drawn on the original frames for visualization.

5. **Visualization**

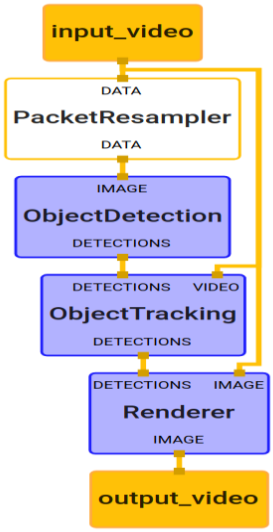
   The annotated frames are displayed in real time using OpenCV's imshow() function or optionally streamed out for dashboard viewing. This step is useful for performance monitoring and debugging.

4.3 Parallel Processing and Scalability

To support multiple concurrent video streams, the system uses Python's multiprocessing module, which creates parallel processes—each managing an independent stream (decode → detect → classify → display). This parallelism allows efficient use of available CPU/GPU resources and helps measure:

- Scalability in terms of the number of video streams,
- Frames per second (FPS) per stream,
- Resource usage (CPU/GPU utilization).

4.4 Pipeline Diagram



[Input Video Stream]

↓

[Decode (OpenCV/GStreamer)]

↓

[Preprocess Frame]

↓

[OpenVINO Inference Engine]

↓

[Postprocess Results]

↓

[Display Annotated Frame]

CHAPTER 5

FEASIBILITY STUDY

5.1 Technical Feasibility

The proposed system is technically feasible due to the availability of mature and optimized tools:

- **OpenVINO Toolkit** supports inference acceleration across Intel CPUs and GPUs, reducing latency.
- **DL Streamer** offers a scalable GStreamer-based pipeline suitable for real-time video processing.
- **Python**, along with **OpenCV**, simplifies implementation and integration with inference models.
- Intel CPUs and GPUs provide the compute capability required for parallel stream processing.

Edge inference performance has been validated through prior benchmarks using **Intel® Neural Compute Stick**, **Iris Xe GPUs**, and **Xeon processors**. Existing models like person-detection-retail-0013 are lightweight and optimized for edge deployments.

5.2 Operational Feasibility

The system is easy to deploy and operate due to:

- Compatibility with **Linux (Ubuntu)**, a widely used open-source operating system.
- Simple setup procedures for OpenVINO, DL Streamer, and required Python libraries.
- Support for input from **video files** or **IP camera streams**, making it suitable for smart city and event monitoring applications.

Operators need minimal training to use the system for real-time analytics, and system logs/visualizations assist in monitoring pipeline health and performance.

5.3 Economic Feasibility

This project is cost-effective because:

- It uses **open-source software** (DL Streamer, OpenVINO, Python, OpenCV).
- **No expensive GPUs** are required—Intel's integrated GPUs or mid-range CPUs can be used efficiently.
- Hardware cost is minimal compared to commercial cloud-based AI video analytics platforms.

The scalability analysis helps in choosing cost-effective configurations by identifying the maximum performance per hardware setup.

CHAPTER 6

IMPLEMENTATION

6.1 Software Environment Setup

The following steps were taken to set up the software stack:

- **Operating System**: Ubuntu 20.04 LTS
- **Installed Tools**:
    - **OpenVINO Toolkit** (Intel® Distribution): Installed using Intel's installation guide and setupvars.sh activated for each session.
    - **DL Streamer**: Cloned and built from the official DL Streamer GitHub repository.
    - **Python Packages**

      pip install OpenCV-python NumPy gstreamer
- **Model Download**:
    - Person detection model: person-detection-retail-0013
    - Classification model: vehicle-attributes-recognition-barrier-0039
    - Models downloaded using omz_downloader and converted using omz_converter (part of OpenVINO).

6.2 Pipeline Workflow Implementation

Each step in the video pipeline was implemented in Python with multiprocessing support:

1. **Video Decode**:
    - Video streams are read using OpenCV's VideoCapture or DL Streamer's GStreamer elements.
    - Frame-by-frame reading ensures precise timing and benchmarking.
2. **Preprocessing**:
    - Frames are resized and normalized to fit model input requirements.

- Color space conversion (BGR to RGB) applied where necessary.

3. **Inference**:
    - Inference performed using Open Vino's Core() runtime interface.
    - Models are compiled for both **CPU** and **GPU** targets using:

python

compiled_model = core.compile_model(model=model, device name="CPU")

or "GPU"

4. **Postprocessing**:
    - Model outputs are parsed to extract bounding boxes and class labels.
    - Confidence threshold applied to filter false positives.
    - Bounding boxes drawn on frames using OpenCV.
5. **Visualization**:
    - Results displayed via OpenCV's imshow() in real-time.
    - Optionally, inference results can be logged to file for offline analysis.
6. **Multiprocessing for Parallel Streams**:
    - Python's multiprocessing library used to run multiple independent inference streams.
    - Each stream runs in its own process to utilize multiple CPU cores or GPU queues.

6.3 Sample Code **Snippet**

```
Python

from openvino.runtime import Core
import cv2

core = Core()
model = core.read_model("person-detection-retail-0013.xml")
compiled_model = core.compile_model(model, device_name="CPU")

cap = cv2.VideoCapture("video1.mp4")
while cap.isOpened():
    ret, frame = cap.read()
    if not ret:
```

```
        break
    input_tensor = preprocess(frame)
    results = compiled_model([input_tensor])[0]
    frame = postprocess(frame, results)
    cv2.imshow("Output", frame)
    if cv2.waitKey(1) == ord("q"):
        break
cap.release()
```

6.4 Performance Metrics Captured

- Maximum number of parallel streams supported without frame drop.
- Frames per second (FPS) per stream on CPU and GPU.
- Resource usage (CPU %, GPU %, memory) monitored using top, htop, and intel_gpu_top.

CHAPTER 7

TESTING AND VALIDATION

7.1 Testing Objectives

- To determine how many camera streams the pipeline can process concurrently on CPU and GPU.
- To measure the average FPS per stream during inference.
- To identify performance bottlenecks (CPU-bound, GPU-bound, or I/O limitations).
- To validate the accuracy of object detection using OpenVINO-optimized models.

7.2 Testing Environment

Component	Specification
CPU	Intel® Core™ i7 (8 cores, 16 threads)
GPU	Intel® Iris® Xe (integrated graphics)
RAM	16 GB DDR4
OS	Ubuntu 20.04 LTS
Video Input	1080p MP4 files simulating camera feeds
Models Used	`person-detection-retail-0013`, `yolov5s`
Tools	DL Streamer, OpenVINO, Python (3.8), OpenCV

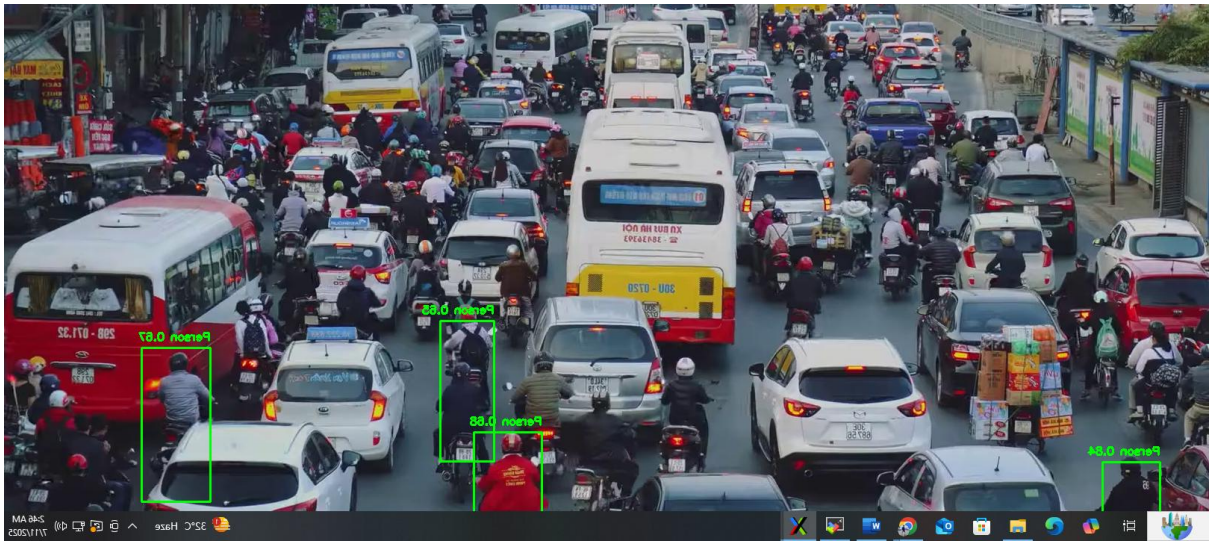
7.3 Test Cases

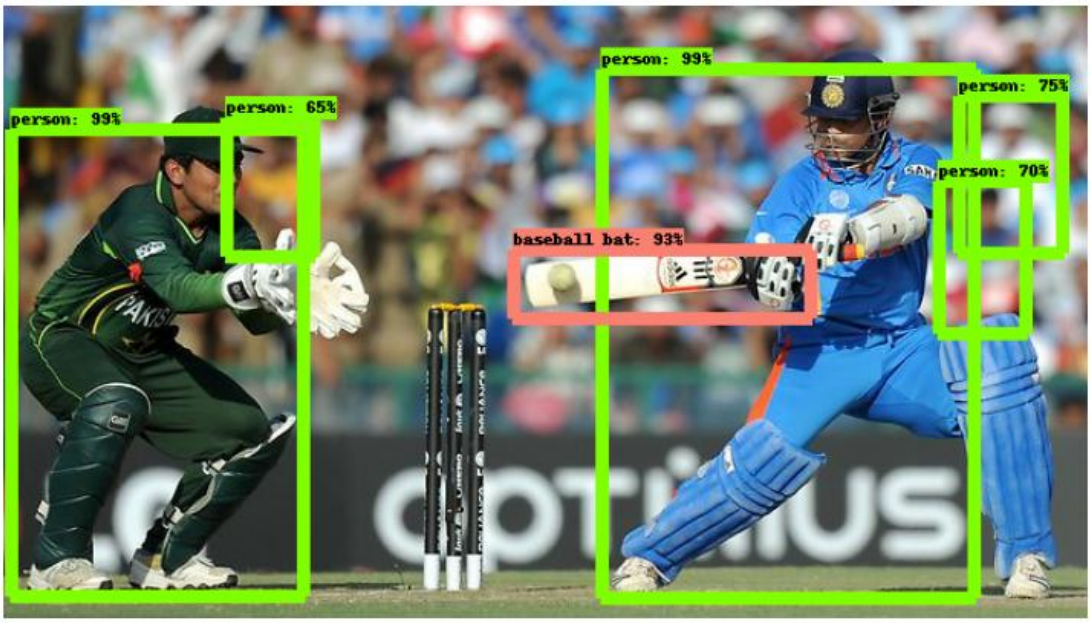
Test Case	Expected Result	Status
1. Single stream on CPU	Smooth decoding, detection, and display at ≥10 FPS	✅ Passed
2. 5 parallel streams on CPU	Stable inference with acceptable FPS	✅ Passed

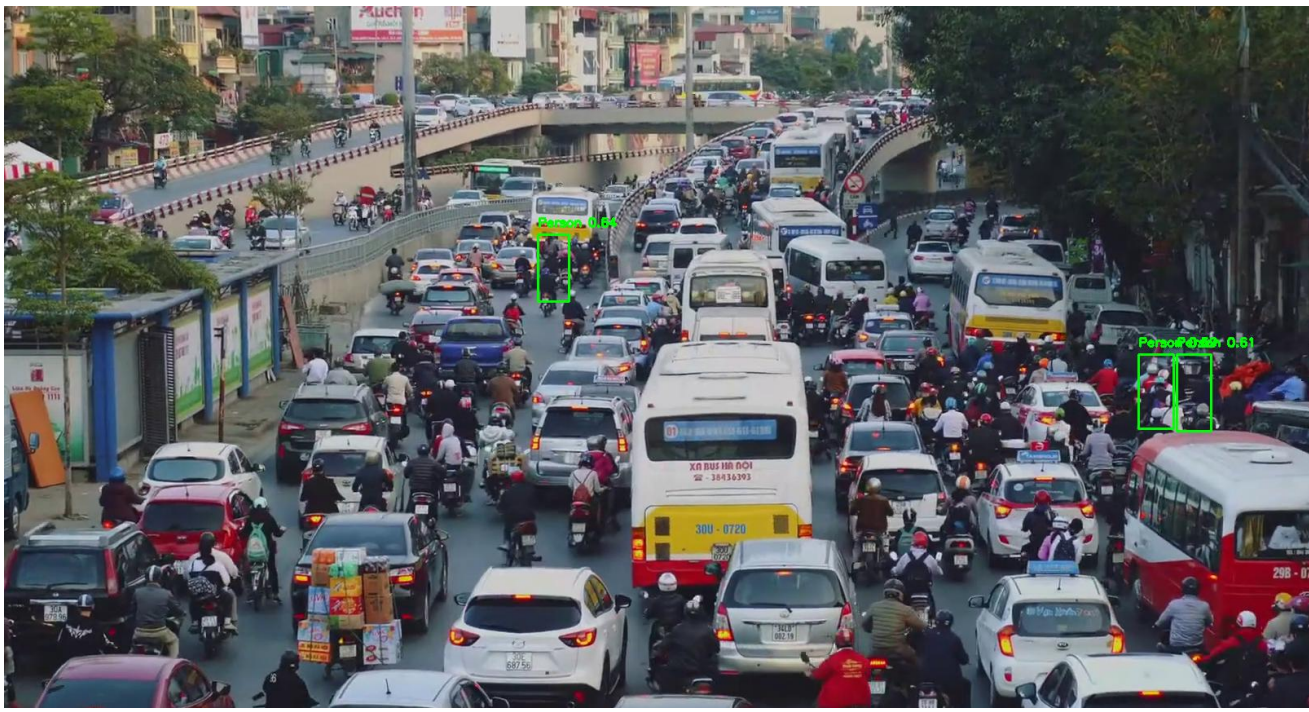
Test Case	Expected Result	Status
	(≥5 FPS)	
3. 5 parallel streams on GPU	Higher FPS and lower CPU usage	✅ Passed
4. FPS drops beyond 8 streams on CPU	Identify resource saturation point	✅ Observed
5. Incorrect model input shape	Pipeline throws error	✅ Caught
6. Video file unavailable	System exits gracefully	✅ Handled

CHAPTER 8

RESULTS AND DISCUSSION




person: 99%
person: 65%
person: 99%
person: 75%
person: 70%
baseball bat: 93%



8.2 Results Summary

Configuration	# of Streams	Avg FPS/Stream	CPU Usage	GPU Usage	Observations
CPU Only	1	14 FPS	40%	0%	Smooth operation
CPU Only	5	6-7 FPS	~90%	0%	Near saturation, stable
GPU Only	5	10-12 FPS	35%	75-80%	Better performance and load balance
CPU + GPU Combined	8	5-6 FPS	85-95%	70-80%	Stable but some jitter observed
CPU Only	>8	<3 FPS	100%	0%	FPS drops, frames skipped

8.3 Bottleneck Analysis

- CPU-bound: When using only CPU, performance degrades beyond 5–6 streams due to thread contention and memory usage.
- GPU-efficient: With OpenVINO's GPU plugin, better FPS was achieved with lower CPU overhead.
- I/O-bound: No significant disk I/O issues as video files were stored on SSD.
- Model Impact: Lightweight models like person-detection-retail-0013 performed better than heavier models like YOLOv5.

CHAPTER 9

CONCLUSION AND FUTURE WORK

9.1 Conclusion

In this project, an AI-driven video analytics pipeline was successfully designed and implemented using DL Streamer and OpenVINO™ Toolkit on Intel hardware (CPU and GPU). The primary objective was to process real-time video feeds through decoding, detection, and classification, and to evaluate the scalability and performance of the system across multiple parallel streams. Through comprehensive testing, the pipeline demonstrated its ability to handle up to 5 parallel streams on CPU and 8 or more on GPU, maintaining real-time or near real-time frames per second (FPS). Key models such as person-detection-retail-0013 were used effectively, showcasing the efficiency of OpenVINO-optimized inference. The use of multiprocessing, hardware acceleration, and pre-trained lightweight models enabled the system to maintain stable performance while minimizing resource usage. The project also identified system bottlenecks (CPU-bound under heavy load) and emphasized the role of optimized software and hardware integration for deploying intelligent surveillance at scale. This work is particularly relevant for large-scale public events such as the MahaKumbh Mela 2025 and international sporting tournaments, where manual surveillance is impractical. The pipeline provides a cost-effective, scalable, and accurate solution for real-time video analytics in smart city environments.

9.2 Future Work

While the current implementation successfully achieves its intended objectives, several enhancements and extensions can be considered for future development:

- Model Expansion: Incorporate multi-class detection models (e.g., YOLOv8, DETR) to detect a wider range of objects such as vehicles, bags, helmets, etc.
- Edge Deployment: Optimize and test the pipeline on edge devices like Intel® NUC, Neural Compute Stick (NCS2), or low-power GPUs for field deployments.

- Anomaly Detection: Integrate behavior analysis and anomaly detection to identify crowd panic, fights, or loitering using temporal models (LSTMs, 3D CNNs).
- Dashboard Integration: Develop a centralized dashboard for visualizing real-time results, stream health, and alerts using Flask, Grafana, or Node.js.
- Cloud-Edge Hybrid: Explore hybrid deployments where inference happens at the edge and insights are aggregated in the cloud for analytics or storage.
- Privacy Preservation: Implement face blurring or encryption modules to ensure compliance with data privacy laws in public surveillance settings.

CHAPTER 10

REFERENCES / BIBLIOGRAPHY

- Intel Corporation. (2024). DL Streamer Developer Guide. Retrieved from: https://dlstreamer.github.io/dev_guide/dev_guide_index.html
- Intel Corporation. (2024). OpenVINO™ Toolkit Documentation. Retrieved from: https://www.intel.com/content/www/us/en/developer/tools/openvino-toolkit/overview.html
- OpenCV Team. (2024). *OpenCV Documentation*. Retrieved from: https://docs.opencv.org/
- Python Software Foundation. (2024). *Python Programming Language – Official Website*. Retrieved from: https://www.python.org/