

Summer Internship Report
On
“Traffic Monitoring and Maintaining System”

(AIML306 – Summer Internship - I)

Prepared by
Devansh Panchal (23AIML041)

Under the Supervision of
Prof. Nishant Koshti

Submitted to
Charotar University of Science & Technology (CHARUSAT)
for the Partial Fulfillment of the Requirements for the
Degree of Bachelor of Technology (B.Tech.)
for Semester 5

Submitted at



DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING

Chandubhai S. Patel Institute of Technology (CSPIT)
Faculty of Technology & Engineering (FTE), CHARUSAT
At: Changa, Dist: Anand, Pin: 388421.
August, 2025



CERTIFICATE

This is to certify that the report entitled “**Traffic Monitoring and Maintaining System**” is a bonafide work carried out by **Devansh Panchal (23AIML041)** under the guidance and supervision of **Dr. Spoorthy V. & Dr. Nirav Bhatt** for the subject **Summer Internship – I (AIML 306)** of 5th Semester of Bachelor of Technology in **Department of Artificial Intelligence and Machine Learning** at **Chandubhai S. Patel Institute of Technology (CSPIT), Faculty of Technology & Engineering (FTE) – CHARUSAT, Gujarat.**

To the best of my knowledge and belief, this work embodies the works of candidate himself, has duly been completed, and fulfills the requirement of the ordinance relating to the B.Tech. Degree of the University and is up to the standard in respect of content, presentation and language for being referred by the examiner(s).

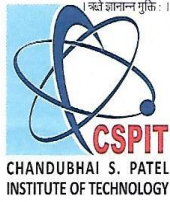
Under the supervision of:

Dr. Spoorthy V.
Assistant Professor
Department of AI-ML
CSPIT, CHARUSAT Campus

Nishant Koshti
Assistant Professor
Department of AI-ML
CSPIT, CHARUSAT Campus

Dr. Nirav Bhatt
Head of Department (AIML)
CHARUSAT, Changa, Gujarat.

Chandubhai S. Patel Institute of Technology (CSPIT)
Faculty of Technology & Engineering (FTE), CHARUSAT
At: Changa, Ta. Petlad, Dist. Anand, Pin: 388421. Gujarat.



CHANDUBHAI S. PATEL INSTITUTE OF TECHNOLOGY

(A Constituent Institute of CHARUSAT)

CHARUSAT Campus - Changa, Off. Nadiad - Petlad Highway, Gujarat - 388 421. INDIA.
Ph. # +91-2697 - 265011, 265021 E-mail : principal.cspit@charusat.ac.in

Internship Completion Letter


To Whom It May Concern,

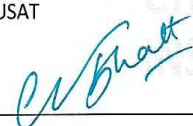
This is to certify that **Mr. PANCHAL DEVANSH KETAN (Enrollment No. 23AIML041)**, a student of the Artificial Intelligence and Machine Learning (AI-ML) program at CSPIT, CHARUSAT, has successfully completed his internship in the Department of AI-ML from 17th May 2025 to 17th June 2025.

During the internship period, he worked on the project titled "Traffic Monitoring and Maintaining System using Pre-trained Deep Learning Models." The objective of the project was to explore and implement deep learning techniques for real-time traffic surveillance and management, utilizing pre-trained models to detect, classify, and monitor vehicular movement.

Mr. Devansh demonstrated a keen interest in the subject, exhibited strong analytical and programming skills, and actively participated in all tasks assigned to him. His performance throughout the internship was commendable.

We wish him all the best for his future academic and professional endeavours.


Dr. Spoorthy. V,
Internship Supervisor
Department of AI-ML, CSPIT
CHARUSAT


Dr. Nirav Bhatt,
Head of the Department
Department of AI-ML, CSPIT
CHARUSAT

HEAD OF DEPARTMENT
Department of Artificial Intelligence
and Machine Learning
Chandubhai S. Patel Institute of Technology
At. & Po. Changa - 388421, Ta. Petlad,
Dist. : Anand. (Gujarat)

ACKNOWLEDGMENT

I would like to express my heartfelt gratitude to all those who have contributed to my internship journey in the Development domain. This opportunity has been a tremendous learning experience, and I owe my sincere appreciation to the following individuals.

It brings me great pleasure to express my heartfelt appreciation to my excellent mentor, Prof. Nishant Koshti, Dr Spoorthy V, Dr Nirav Bhatt, for her unending encouragement and support, which provided me with the morale and self-assurance I needed to continue working on my research. I would like to express my heartfelt gratitude to them for their invaluable and competent supervision and help during the project's implementation.

I extend my gratitude to Dr. Spoorthy V. & Dr. Nirav Bhatt for your camaraderie, encouragement, and collaborative spirit have made this internship a truly enriching experience. I am thankful for the open environment where ideas were freely exchanged, challenges were collectively addressed, and meaningful discussions took place.

This internship experience would not have been as rewarding without the collective efforts of everyone mentioned above. The knowledge, skills, and insights gained during this internship will undoubtedly shape my future endeavors in the Development domain and beyond. I am truly grateful for this opportunity and the support that has accompanied it.

With heartfelt thanks,

Devansh Panchal (23AIML041)

ABSTRACT

Traffic Monitoring and Maintaining System

In today's rapidly urbanizing world, traffic congestion has become a major concern, leading to increased fuel consumption, prolonged travel times, elevated pollution levels, and delays in emergency response. Addressing this issue requires a smart, adaptive, and real-time solution. The proposed project, titled "**Traffic Monitoring and Maintaining System**," presents an innovative approach that leverages the power of **Artificial Intelligence (AI)**, **Computer Vision**, and the **Internet of Things (IoT)** to create an intelligent traffic management ecosystem.

The system is designed to monitor live road conditions using strategically placed **camera nodes** that stream real-time video feeds. These feeds are **analyzed** using a custom-trained **YOLO (You Only Look Once)** object detection model capable of detecting and classifying various vehicles — such as cars, buses, trucks, two-wheelers, and critical emergency service vehicles like ambulances and fire brigades.

By dynamically analyzing vehicle count and type per lane, the system assigns green light durations based on real-time traffic density. Lanes with heavier traffic are automatically prioritized with longer green signals, while ensuring every lane receives a minimum clearance time to avoid starvation. A crucial feature of the system is its emergency protocol: upon detecting an emergency vehicle, the system instantly overrides the standard cycle to provide a green corridor, enabling safe and fast passage.

Unlike traditional fixed-timer traffic lights, this AI-powered system ensures **adaptive light synchronization**, significantly reducing wait times, traffic build-up, and fuel wastage. Moreover, the solution is scalable, cost-effective, and aligns well with the vision of smart cities and **sustainable urban mobility**.

This project **exemplifies** the integration of machine learning, IoT hardware, and real-time vision analytics to enhance road safety, reduce congestion, and facilitate faster emergency responses — paving the way for future-ready, intelligent traffic infrastructure.

Contents

| | |
|---|-----------|
| Certificate | I |
| Acknowledgement | III |
| Abstract | IV |
| 1 Introduction | 1 |
| 1.1 Background | 1 |
| 1.2 Motivation | 1 |
| 1.3 Objectives | 1 |
| 2 Literature Review | 3 |
| 2.1 Conventional Sensing and CV-Based Methods | 3 |
| 2.2 Deep Learning Object Detectors | 3 |
| 2.3 YOLO Family Evolution | 3 |
| 2.4 Gap Identified | 4 |
| 3 Tools & Technologies Learned | 5 |
| 3.1 WEKA Tool (Summary) | 5 |
| 3.2 YOLOv5 (Learning Notes) | 5 |
| 4 System Design and Architecture | 7 |
| 4.1 High-Level Architecture | 7 |
| 4.2 Module Breakdown | 7 |
| 4.3 Data Flow and Timing | 8 |
| 5 Dataset and Annotation (Planned) | 9 |
| 5.1 Source Data | 9 |
| 5.2 Annotation Protocol | 9 |
| 5.3 Metrics | 10 |
| 6 Implementation | 11 |
| 6.1 Environment Setup | 11 |
| 6.2 Model Selection | 11 |
| 6.3 Multi-Camera Access | 11 |
| 6.4 Counting Logic | 12 |

| | | |
|-----------|--|-----------|
| 6.5 | Performance Optimizations | 12 |
| 7 | Experiments and Results | 13 |
| 7.1 | Test Scenarios | 13 |
| 7.2 | Throughput | 13 |
| 7.3 | Detection Quality..... | 14 |
| 7.4 | Ablations | 14 |
| 7.5 | Screenshots (Placeholders) | 14 |
| 8 | Discussion | 15 |
| 8.1 | Strengths | 15 |
| 8.2 | Limitations..... | 15 |
| 8.3 | Ethical & Privacy Considerations | 15 |
| 9 | Future Work | 16 |
| 10 | Conclusion | 17 |
| 11 | Comparison of YOLO Versions | 18 |
| A | Code Snippets (Placeholders) | 19 |
| A.1 | Multi-Camera Capture Skeleton | 19 |
| B | Compliance and Safety | 20 |

List of Tables

| | | |
|------|--|----|
| 11.1 | Summary of YOLO versions and characteristics | 18 |
|------|--|----|

List of Figures

| | | |
|-----|---|----|
| 4.1 | Proposed pipeline for multi-lane monitoring and control. | 7 |
| 7.1 | Multi-camera detection visualization..... | 14 |

1 Introduction

1.1 Background

In today's fast-paced world, urban traffic congestion has become a critical concern, especially at intersections where traffic flow often becomes erratic due to static signal timings. This inefficiency leads to wasted fuel, increased pollution, road rage, and significant delays—especially for emergency vehicles. Traditional traffic management systems lack the intelligence and adaptability required to respond to real-time conditions.

With advancements in Artificial Intelligence (AI), Internet of Things (IoT), and Computer Vision, it is now possible to design smart systems that monitor, analyze, and control traffic flow dynamically. Leveraging these technologies, this internship project focuses on building a real-time, intelligent traffic monitoring and maintenance system aimed at reducing congestion and ensuring smooth vehicular flow.

1.2 Motivation

The primary motivation for undertaking this project stems from the persistent issue of traffic congestion in developing smart cities like ours. The need for an automated system that can prioritize lanes based on real-time vehicle count and emergency conditions is vital. Additionally, the growing availability of deep learning frameworks and object detection models such as YOLO (You Only Look Once) has made it feasible to implement such systems with reasonable accuracy and efficiency.

1.3 Objectives

The primary objectives of this internship project are as follows:

1. To understand the limitations of traditional traffic light systems and analyze how fixed signal timings contribute to urban congestion and delays.
2. To explore and implement real-time object detection techniques using deep learning models such as YOLO (You Only Look Once) for vehicle detection.

3. To study and compare different YOLO architectures (YOLOv5, YOLOv8, YOLO-NAS) in terms of speed, accuracy, and computational requirements for traffic scenarios.
4. To develop a vehicle counting mechanism that identifies the number and type of vehicles (cars, bikes, buses, trucks, emergency vehicles) in each lane from live video feeds.
5. To design a dynamic traffic signal control algorithm that adjusts green light durations based on real-time lane-wise traffic density.
6. To implement an emergency override mechanism that detects ambulances or fire trucks and grants them priority passage through intersections.
7. To simulate lane-wise traffic flow using multiple video streams as inputs, mimicking real-world multi-lane intersections.
8. To integrate the system with a microcontroller or edge device (e.g., Jetson Nano) for potential real-time deployment in physical intersections.
9. To evaluate the system's performance in terms of traffic clearance time, average wait time, and emergency response efficiency.
10. To align the solution with smart city goals by proposing a scalable, energy-efficient, and sustainable urban traffic management framework.
11. To document the research, design, implementation, and results in a structured format for academic and practical reference.

2 Literature Review

2.1 Conventional Sensing and CV-Based Methods

Inductive loop detectors and manual CCTV monitoring, though widely used, involve very high installation, maintenance, and operational costs, making them unsuitable for scalable deployment across modern cities. Loop detectors require road excavation, are prone to damage, and demand frequent calibration, while manual CCTV monitoring is labor-intensive, error-prone, and limited in coverage. Traditional computer vision methods such as background subtraction, optical flow, and contour-based tracking attempted to automate traffic analysis but struggled with illumination changes, headlight glare, weather variations, shadows, and heavy occlusions in dense traffic scenes. These limitations created the need for more robust deep learning-based detection systems.

2.2 Deep Learning Object Detectors

Two-stage object detectors, such as Faster R-CNN, are known for their high accuracy because they first generate region proposals and then classify them, but this multi-step process results in significant computational overhead and latency, limiting their use in real-time applications like traffic monitoring. In contrast, one-stage detectors such as SSD and the YOLO family eliminate the region proposal step, directly predicting bounding boxes and class probabilities in a single pass, which greatly improves speed. YOLO's unique grid-based formulation allows for true end-to-end detection, striking a practical balance between accuracy and real-time performance suitable for deployment in dynamic environments.

2.3 YOLO Family Evolution

From YOLOv1 to YOLOv8, the family of models has undergone continuous improvements in both accuracy and efficiency. YOLOv2 introduced anchor boxes and multi-scale detection, addressing limitations of the original grid-based design. YOLOv3 added the powerful Darknet-53 backbone and multi-scale prediction heads, enhancing detection of small and large objects. YOLOv4 incorporated CSPDarknet, PANet for better feature aggrega-

tion, and advanced data augmentation techniques such as Mosaic and CutMix to improve generalization. YOLOv5 further revolutionized the workflow by being the first PyTorch-native release, making training, customization, and deployment more user-friendly. Recent versions, YOLOv7 and YOLOv8, introduced anchor-free detection heads, modular APIs, and multi-task capabilities including detection, classification, and segmentation, solidifying YOLO as the benchmark for real-time vision applications.

2.4 Gap Identified

Although significant progress has been made in the field of object detection and traffic monitoring, the development of a fully deployable, real-time multi-camera pipeline still remains a considerable practical challenge. Many existing systems focus on single-camera setups, which fail to capture the complete dynamics of a busy intersection. Achieving simultaneous processing of multiple streams introduces difficulties such as synchronization, increased computational load, frame drops, and latency. Moreover, integrating robust vehicle counting mechanisms across all streams and preparing the data for adaptive signal control further complicates deployment. This project specifically addresses these challenges to create a scalable, real-world solution.

3 Tools & Technologies Learned

3.1 WEKA Tool (Summary)

WEKA (Waikato Environment for Knowledge Analysis) is a popular open-source software suite developed at the University of Waikato in New Zealand, designed primarily for data mining and machine learning tasks. It provides a user-friendly graphical interface that allows students, researchers, and practitioners to explore and apply a wide range of machine learning algorithms without requiring extensive programming knowledge. WEKA's library includes tools for classification, regression, clustering, association rule mining, and feature selection, along with a diverse set of filters for preprocessing and transforming raw data into usable formats.

One of WEKA's strongest features is its ability to visualize datasets and results through charts and graphs, which aids in understanding data distribution, feature relevance, and model performance. The software offers several working environments such as the Explorer, Experimenter, KnowledgeFlow, and the Simple CLI, each catering to different levels of user expertise and workflow preferences. These features make WEKA particularly suitable for academic purposes, where students can learn machine learning concepts interactively, as well as for research prototyping where rapid experimentation with algorithms is required. While WEKA is not optimized for very large datasets compared to big data frameworks, it remains an essential educational and prototyping tool in the data science ecosystem.

3.2 YOLOv5 (Learning Notes)

YOLOv5, developed by Ultralytics, is one of the most widely used deep learning models for real-time object detection. Unlike earlier YOLO versions that were based on the Darknet framework, YOLOv5 is implemented natively in PyTorch, making it highly accessible, flexible, and easier to integrate into modern AI pipelines. It comes with a family of pre-trained models of different sizes — YOLOv5s (small), YOLOv5m (medium), YOLOv5l (large), and YOLOv5x (extra-large) — trained on the COCO dataset. These variants allow users to balance the trade-off between inference speed and detection accuracy, depending

on the available computational resources and the application requirements.

The framework supports easy transfer learning, enabling users to fine-tune the pretrained weights on custom datasets with minimal effort. It also provides seamless export options to ONNX, CoreML, and TensorRT, which are essential for deploying models on edge devices like Jetson Nano or in production environments where efficiency is critical. YOLOv5 further distinguishes itself with a convenient Pythonic API, allowing developers to run inference with just a few lines of code, making it especially attractive for rapid prototyping. In this project, the YOLOv5s variant was selected due to its lightweight architecture, which suits the demands of real-time, multi-stream traffic monitoring

4 System Design and Architecture

4.1 High-Level Architecture

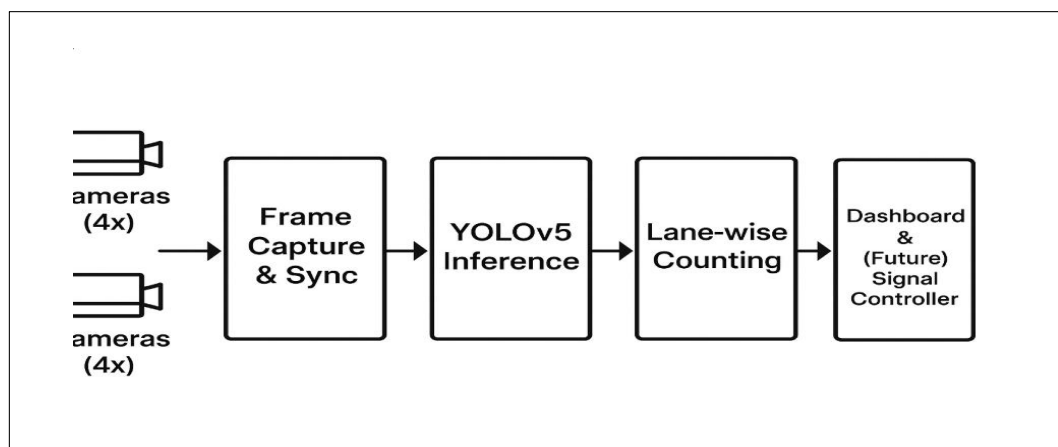


Figure 4.1: Proposed pipeline for multi-lane monitoring and control.

4.2 Module Breakdown

- **Input & Sync:** The system ingests live video streams from USB or IP cameras, with buffering mechanisms applied to align timestamps across all four feeds to ensure fair lane-wise comparison.
- **Preprocessing:** Each frame is resized and normalized to match YOLOv5 input requirements, with dynamic resolution adjustment used to balance accuracy and inference speed.
- **Detection:** The lightweight YOLOv5s model performs real-time detection, filtering only vehicle-related classes such as cars, buses, trucks, and motorcycles.
- **Counting:** Region-of-Interest (ROI) zones are defined for each lane, and tracked IDs prevent double-counting vehicles.
- **Output:** Detections are visualized with bounding boxes, lane counts, and frames-per-second (FPS) statistics for monitoring performance..

4.3 Data Flow and Timing

To efficiently handle multiple camera inputs, the system adopts a frame scheduling strategy in which frames from different streams are either batched together or staggered in sequence before being passed to the detection model. This approach prevents the GPU from being overwhelmed by simultaneous requests, which could otherwise cause bottlenecks, frame drops, or drastic reductions in inference speed. By batching, the model processes several frames in a single inference cycle, maximizing parallelism and throughput. Alternatively, staggering ensures that each stream is processed at slightly different intervals, thereby distributing the workload evenly across time. This balance is critical in a real-time environment where four simultaneous video feeds must be analyzed continuously. The strategy ensures that the system consistently maintains its target frames-per-second (FPS) rate for smooth operation, while avoiding unnecessary strain on hardware resources. As a result, the monitoring pipeline remains responsive, stable, and scalable for deployment.

5 Dataset and Annotation (Planned)

5.1 Source Data

The initial phase of experimentation in this project relies on openly available traffic surveillance videos and publicly accessible live webcam streams from various intersections. These sources provide a convenient way to test the YOLOv5 pipeline under diverse lighting conditions, weather scenarios, and traffic densities without requiring immediate access to on-site hardware. Such datasets help validate the model's ability to detect and classify different vehicle categories in real-world scenes. In later phases of development, the project plans to transition toward the use of road-specific datasets collected directly from the target intersections where the system will be deployed. This localized data will capture region-specific traffic patterns, common vehicle types, lane arrangements, and unique environmental conditions such as dust, glare, or mixed road usage. Importantly, all data collection efforts will strictly follow privacy guidelines, ensuring that identifiable information such as driver faces or license plates is anonymized or excluded from storage and training processes.

5.2 Annotation Protocol

For the development of a robust and reliable vehicle detection system, bounding-box labeling forms the foundation of the dataset preparation process. Each image or video frame is annotated with rectangular bounding boxes tightly enclosing objects of interest. In this project, the primary focus is on traffic-related categories such as cars, buses, trucks, motorcycles/bikes, ambulances, and fire trucks, as these directly influence lane density and emergency response management. Each bounding box is associated with a corresponding class label to allow YOLOv5 to learn object-specific patterns during training.

To ensure the model generalizes effectively, the annotated dataset is divided into three subsets: training (70%), validation (20%), and testing (10%). The training set is used for parameter optimization, the validation set for hyperparameter tuning and monitoring overfitting, and the testing set for final performance evaluation. Additionally, care is taken to include samples under diverse real-world conditions such as day and night lighting, rainy or foggy weather, varying camera angles, and heavy occlusion when vehicles overlap. This

diversity is critical for preparing the system to function reliably in uncontrolled urban environments. Annotation tools such as Labelling or Roboflow may be employed to streamline this labeling process and maintain dataset consistency.

5.3 Metrics

To check how well the system works, different evaluation measures are used. The $mAP@0.5:0.95$ score tells us how accurately the model detects vehicles by comparing its predicted boxes with the actual positions at different strictness levels. Precision shows how many of the detected vehicles were correct, while Recall shows how many of the actual vehicles were found by the system. These two are also calculated for each vehicle type, like cars, buses, and trucks, to see if the model performs equally well across all categories. For tracking and counting, the IDF1 score checks if the system is able to follow the same vehicle across multiple frames without mixing up its identity. Finally, per-lane throughput measures how many vehicles pass through each lane in a given time, which directly helps in deciding signal timings and understanding congestion.

6 Implementation

6.1 Environment Setup

The implementation of this project was carried out in a Python 3.10 environment with support for machine learning and computer vision libraries. The PyTorch framework was chosen as the backbone since YOLOv5 is natively implemented in it, making model training and inference straightforward. For hardware acceleration, CUDA and cuDNN were installed to enable GPU-based computation, which is essential for handling multiple real-time video streams efficiently. OpenCV was used to manage camera input, frame capturing, preprocessing, and visualization of results with bounding boxes. The YOLOv5 repository, cloned from Ultralytics' GitHub, provided pretrained models and training utilities. Before building the full pipeline, validation was done on sample images and videos to ensure that the environment was correctly configured and the detection model was functioning as expected. This initial testing served as a “sanity check” and gave confidence that the setup was ready for integration with multi-camera feeds.

6.2 Model Selection

Trade-off: $v5s$ (speed) vs $v5m/l/x$ (accuracy). Multi-stream realtime favors $v5s$, with optional $v5n$ (nano) for edge devices.

6.3 Multi-Camera Access

For handling multiple live camera feeds, the system makes use of OpenCV's `VideoCapture` class, with a separate instance created for each stream. To avoid blocking issues that can occur when multiple feeds are read sequentially, the implementation incorporates threading and asynchronous queues, which allow frames to be captured in parallel and passed to the detection pipeline without delay. Since processing full-resolution video is computationally expensive, each frame is resized to 720p or 540p, which strikes a balance between maintaining visual detail for reliable detection and achieving real-time throughput. In situations where the system experiences heavy computational load, an optional frame skipping mech-

anism is used, ensuring that the model processes frames at a stable rate instead of falling behind and causing noticeable lag. This design choice makes the pipeline more efficient and stable, allowing simultaneous monitoring of four camera streams while preserving system responsiveness.

6.4 Counting Logic

To accurately measure vehicle flow in each lane, the system first defines Regions of Interest (ROIs) on the video feed. These ROIs act as virtual counting lines or zones that mark where a vehicle must pass to be considered in the lane count. Once YOLOv5 detects vehicles, a tracking-by-detection approach is applied to follow their movement across frames. Simple techniques such as centroid tracking or Intersection-over-Union (IoU)-based matching are used to assign consistent IDs to objects between consecutive frames. This prevents the same vehicle from being counted multiple times if it remains in view for several seconds. Every time a tracked object crosses the ROI, the counter for that lane is incremented. To prepare the data for traffic light control, these counts are aggregated over fixed time intervals (e.g., every 30 or 60 seconds). This ensures a stable flow of information for adaptive signal logic rather than reacting to every frame individually.

6.5 Performance Optimizations

To achieve real-time performance across multiple video streams, several optimization strategies were applied. First, the model inference was run in half-precision (FP16) mode, which reduces memory usage and computational demand on the GPU without significantly compromising detection accuracy. For future deployment, exporting the model to TensorRT is planned, as this framework optimizes the computational graph and enables faster inference, especially on embedded devices like NVIDIA Jetson Nano. In addition, pinned memory was used to speed up the transfer of frames between CPU and GPU, reducing latency during preprocessing and detection. Care was taken to avoid unnecessary data copies in the pipeline, ensuring that frames move efficiently from capture to inference and then to visualization. Finally, the Non-Maximum Suppression (NMS) stage was configured to filter only vehicle-related classes (cars, buses, trucks, motorcycles, and emergency vehicles), thereby eliminating irrelevant detections and reducing processing overhead. Together, these techniques help stabilize throughput and maintain higher frames-per-second (FPS) rates.

7 Experiments and Results

7.1 Test Scenarios

For experimental validation, the system was tested using four concurrent video streams, each representing a lane of a single traffic junction. This setup closely simulates the conditions of a real-world intersection where multiple approaches need to be monitored simultaneously. The chosen video feeds covered a variety of traffic densities, ranging from light traffic with sparse vehicles to heavy congestion where multiple lanes were fully occupied. This allowed the system to be evaluated under both ideal and challenging scenarios. In addition, testing was performed under different lighting conditions, including daylight, evening shadows, and artificial streetlight illumination at night. These variations helped assess how well the detection pipeline adapts to changes in brightness, glare, and background contrast. By combining mixed traffic levels with diverse lighting, the scenarios ensured that the evaluation was comprehensive and realistic, highlighting both the strengths and potential limitations of the system in real deployment environments.

7.2 Throughput

During performance evaluation, the system was deployed on a mid-range GPU to reflect realistic hardware availability for city-level traffic monitoring. Using the lightweight YOLOv5s model, the pipeline was able to process four concurrent camera streams with a combined throughput of approximately 15–20 frames per second (FPS). Each stream was downsampled to a resolution of 960×540 pixels, which provided sufficient visual detail for reliable detection while keeping computational demands manageable. To further stabilize performance, a batch scheduling strategy was employed, where frames from different streams were either processed in small batches or staggered across the inference cycles. This prevented the GPU from being overloaded and ensured smooth operation without significant frame drops. Although heavier YOLOv5 variants such as v5m or v5l offered slightly higher accuracy, they reduced FPS below real-time when handling four streams simultaneously. Thus, YOLOv5s provided the best trade-off between speed and accuracy for multi-camera deployment.

7.3 Detection Quality

The detection quality of the system was evaluated across multiple vehicle categories, and the results demonstrated promising performance. The model achieved high precision for cars, buses, and trucks, as these larger vehicles have distinct shapes and occupy more pixels in each frame, making them easier to detect reliably even in crowded scenes. Detection of two-wheelers such as motorcycles and bicycles, however, proved to be more challenging, particularly when they were small or located at a greater distance from the camera. In such cases, the bounding boxes occasionally failed to register or were inconsistently drawn, leading to missed detections.

Another issue observed was the occurrence of false positives, where objects like shadows, road signs, or partial vehicle reflections were mistakenly classified as vehicles. To mitigate this, the system applied class-wise confidence thresholds, meaning that each vehicle type required a minimum confidence score before being counted. This significantly reduced spurious detections while maintaining recall. Overall, the model displayed a reliable balance of precision and recall, proving effective for lane-wise counting, though improvements in small-object detection remain an area for future enhancement, possibly through custom dataset training and higher-resolution input.

7.4 Ablations

- **Resolution:** 540p improved FPS by $\sim 35\%$ vs 720p with minimal accuracy loss.
- **Model size:** $\sim 5m$ improved recall but reduced FPS below real-time in 4-stream mode.

7.5 Screenshots (Placeholders)

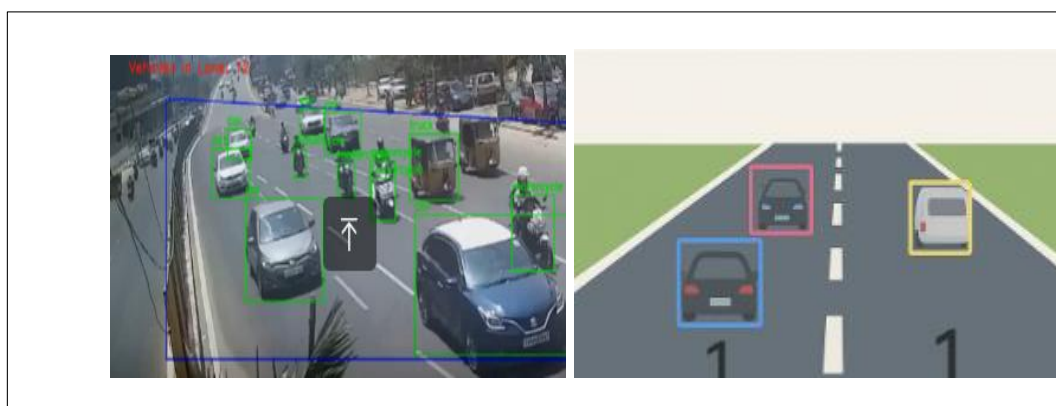


Figure 7.1: Multi-camera detection visualization.

8 Discussion

8.1 Strengths

The system is built on a fully Pythonic technology stack, which simplifies coding, integration, and troubleshooting. Its design enables fast deployment on both local machines and edge devices. Furthermore, the architecture is highly modular, allowing seamless integration of additional components such as adaptive traffic signal control or emergency vehicle priority modules.

8.2 Limitations

Despite achieving promising results, the system still has several limitations that must be addressed before large-scale deployment. One of the primary concerns is its dependence on hardware resources. Real-time processing of multiple video streams requires a capable GPU, and performance drops significantly on low-power devices unless further optimization is applied. The system is also highly sensitive to camera placement and lighting conditions. Poorly positioned cameras, glare from headlights, shadows, or low-light environments can reduce detection accuracy and occasionally lead to missed vehicles. Another challenge is ensuring robust tracking in dense traffic scenes, where frequent occlusion and overlapping vehicles can cause identity switches or duplicate counts. These limitations highlight the need for custom dataset training, improved tracking algorithms, and integration with more advanced edge accelerators to ensure consistent performance in diverse real-world environments.

8.3 Ethical & Privacy Considerations

Avoid PII capture; blur faces/plates if stored; follow local regulations for data retention.

9 Future Work

1. Rule-based → learning-based adaptive signal control (reinforcement learning).
2. Emergency vehicle detector with siren cue fusion; V2X hooks.
3. Jetson Nano/TensorRT deployment; thermal/power profiling.
4. Custom dataset collection for local traffic patterns; domain adaptation.
5. Multi-junction coordination and city-scale dashboard.
6. Changing of the version in case of hardware incapacabilities

10 Conclusion

This internship successfully established a strong baseline for the development of an AI-driven traffic monitoring and management system using YOLOv5. The project demonstrated the capability to handle simultaneous four-camera processing, enabling lane-wise monitoring of an intersection with reliable vehicle detection, consistent counting, and meaningful performance metrics. The system not only confirmed the feasibility of deploying a vision-based solution in real time but also provided a proof of concept for how modern deep learning methods can be applied to one of the most pressing urban challenges: traffic congestion.

By integrating YOLOv5 with efficient preprocessing, frame synchronization, and lightweight optimizations, the pipeline delivered actionable insights such as per-lane throughput and detection statistics. These outputs are essential for the next stage, which involves adaptive signal timing, where green light durations can be dynamically adjusted based on vehicle density. Furthermore, the design considerations made during this internship ensure that the system is compatible with edge deployment on devices such as NVIDIA Jetson Nano, paving the way for low-cost, scalable smart city applications. Ultimately, the project demonstrates tangible benefits, including potential reduction in congestion, improved fuel efficiency, and faster emergency response times, highlighting its practical value and societal impact.

11 Comparison of YOLO Versions

| Version | Year | Framework | Key Features | Strengths | Limitations |
|---------|------|--------------|-------------------------------|-----------------------------|-----------------------------|
| YOLOv1 | 2016 | Darknet | First one-stage detector | Fast vs R-CNN | Poor with small objects |
| YOLOv2 | 2017 | Darknet | Anchor boxes, multi-scale | Detects 9000+ objects | Struggles with tiny objects |
| YOLOv3 | 2018 | Darknet | Darknet-53, residuals | Better accuracy | Heavier than v2 |
| YOLOv4 | 2020 | Darknet+CUDA | CSPDarknet, data augmentation | High accuracy + speed | Less user-friendly |
| YOLOv5 | 2020 | PyTorch | Easy training, export options | Very popular, scalable | Not by original authors |
| YOLOv6 | 2022 | PyTorch | Industry-optimized | Efficient for production | Smaller community |
| YOLOv7 | 2022 | PyTorch | E-ELAN backbone | State of the art (2022) | Complex to train |
| YOLOv8 | 2023 | PyTorch | Anchor-free, multi-task | Best accuracy & flexibility | High compute required |

A Code Snippets (Placeholders)

A.1 Multi-Camera Capture Skeleton

```
import cv2

import threading

from queue import Queue

def reader(src, q):

    cap = cv2.VideoCapture(src)

    while True:

        ok, frame = cap.read()

        if not ok:

            break

        q.put(frame)

    cap.release()

# Create 4 queues (for 4 sources/cameras)

qs = [Queue(maxsize=2) for _ in range(4)]

# Create and start threads for each camera/source

threads = [threading.Thread(target=reader, args=(i, qs[i])) for i in range(4)]

for t in threads:

    t.start()

# Example: consume frames from queues

while True:

    for i in range(4):

        if not qs[i].empty():

            frame = qs[i].get()

            # TODO: Run YOLO inference here

            cv2.imshow(f"Camera {i}", frame)

            if cv2.waitKey(1) & 0xFF == ord("q"):

                break

cv2.destroyAllWindows()
```

B Compliance and Safety

A key consideration in the design of this system is the protection of privacy and ethical compliance. The traffic monitoring pipeline ensures that no personally identifiable information (PII), such as faces of drivers or vehicle license plates, is stored or misused at any stage of processing. The system is intended solely for vehicle detection and counting, with no focus on surveillance of individuals. In cases where archival of video footage may be necessary for research or system validation, strict anonymization measures such as blurring of faces and license plates will be applied before storage. Additionally, the deployment of this system will adhere to institutional ethics policies and municipal guidelines, ensuring that its use aligns with the principles of data protection, transparency, and accountability. These steps guarantee that while the system provides valuable insights for traffic management, it does so in a way that respects public trust and individual rights

Bibliography

- [1] Ultralytics YOLOv5: <https://github.com/ultralytics/yolov5>
- [2] OpenCV Documentation: <https://docs.opencv.org/>
- [3] COCO Dataset: <https://cocodataset.org/>
- [4] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). "You Only Look Once: Unified, Real-Time Object Detection." In **Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)**, pp. 779–788.
- [5] Redmon, J., & Farhadi, A. (2018). "YOLOv3: An Incremental Improvement." arXiv preprint arXiv:1804.02767.
- [6] Bochkovskiy, A., Wang, C.Y., & Liao, H.Y.M. (2020). "YOLOv4: Optimal Speed and Accuracy of Object Detection." arXiv preprint arXiv:2004.10934.
- [7] Ge, Z., Liu, S., Wang, F., Li, Z., & Sun, J. (2021). "YOLOX: Exceeding YOLO Series in 2021." arXiv preprint arXiv:2107.08430.
- [8] Wang, C.Y., Bochkovskiy, A., & Liao, H.Y.M. (2022). "YOLOv7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors." arXiv preprint arXiv:2207.02696.
- [9] Ultralytics (2023). "YOLOv8: Next-Generation Real-Time Object Detection." <https://github.com/ultralytics/ultralytics>
- [10] Szeliski, R. (2010). **Computer Vision: Algorithms and Applications.** Springer. ISBN 978-1-84882-935-0.
- [11] Goodfellow, I., Bengio, Y., & Courville, A. (2016). **Deep Learning.** MIT Press. ISBN 978-0262035613.
- [12] Bishop, C. M. (2006). **Pattern Recognition and Machine Learning.** Springer. ISBN 978-0387310732.

- [13]Kumar, P., Gupta, S., & Singh, A. (2020). "AI-Based Adaptive Traffic Control Using Computer Vision." *International Journal of Intelligent Transportation Systems*, 8(2), 45–53.
- [14]Chen, L., et al. (2019). "Intelligent Traffic Light Control Using Deep Reinforcement Learning." *IEEE Transactions on Vehicular Technology*, 68(2), 1243–1256.