Visvesvaraya Technological University

Jnana Sangama, Belagavi - 590018

A Project Work Phase-2 (BAD786)
Report On

"AI POWERED TRAFFIC SIGNAL PROCESSING"

Submitted in partial fulfilment of the requirement for the award of the degree

of

BACHELOR OF ENGINEERING

in

ARTIFICIAL INTELLIGENCE & DATA SCIENCE

Submitted by

HARSHITHA V	**1KG22AD020**
M JOSHNA	**1KG22AD034**
MOKSHITHA S THOTA	**1KG22AD039**
R ALEKHYA	**1KG22AD045**

Under the guidance of
Mrs. P S GEETHA
Assistant Professor

DEPARTMENT OF ARTIFICIAL INTELLIGENCE & DATA SCIENCE
K. S. School of Engineering and Management
#15, Mallasandra, off. Kanakapura Road, Bengaluru – 560109

2025 - 2026

K. S. School of Engineering and Management

#15, Mallasandra, off. Kanakapura Road, Bengaluru - 560109

Department of Artificial Intelligence & Data Science

CERTIFICATE

Certified that the Project Work Phase – II (BAD786) entitled **"AI POWERED TRAFFIC SIGNAL PROCESSING"** is a bonafide work carried out by:

HARSHITHA V	**1KG22AD020**
M JOSHNA	**1KG22AD034**
MOKSHITHA S THOTA	**1KG22AD039**
R ALEKHYA	**1KG22AD045**

in partial fulfilment for VII semester B.E., Project Work Phase-II (BAD786) in the branch of Artificial Intelligence & Data Science prescribed by **Visvesvaraya Technological University, Belagavi** during the academic year 2025-2026. It is certified that all the corrections and suggestions indicated for internal assessment have been incorporated. The Project Work Phase-II (BAD786) report has been approved as it satisfies the academic requirements in respect of project work prescribed for the Bachelor of Engineering degree.

……	…….	…….
Signature of the Guide	**Signature of the HOD**	**Signature of the Principal**
Mrs. P S Geetha	Dr. Manjunath T K	Dr. B. Balaji
Assistant Professor	Professor & Head	(I/C) Principal, KSSEM

University Examiners

Name of the Examiners **Signature with date**

1.

2.

DECLARATION

We, the undersigned students of 7th semester, Artificial Intelligence & Data Science, KSSEM, declare that our Project Work Phase- II entitled **"AI POWERED TRAFFIC SIGNAL PROCESSING"** is a bonafide work of ours. Our project is neither a copy nor by means a modification of any other engineering project.

We also declare that this project was not entitled for submission to any other university in the past and shall remain the only submission made and will not be submitted by us to any other university in the future.

Place:
Date:

Name and USN		**Signature**
HARSHITHA V	**(1KG22AD020)**	**…….**
M JOSHNA	**(1KG22AD034)**	**…….**
MOKSHITHA S THOTA	**(1KG22AD039)**	**…….**
R ALEKHYA	**(1KG22AD045)**	**…….**

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task will be incomplete without the mention of the individuals, we are greatly indebted to, who through guidance and providing facilities have served as a beacon of light and crowned our efforts with success.

We would like to express our gratitude to our **MANAGEMENT,** K.S. School of Engineering and Management, Bengaluru, for providing a very good infrastructure and all the kindness forwarded to us in carrying out this project work in college.

We would like to express our gratitude to **Dr. B. Balaji, (I/C) Principal**, K.S. School of Engineering and Management, Bengaluru, for his valuable guidance.

We like to extend our gratitude to **Dr. Manjunath T K**, **Professor and Head**, Department of Artificial Intelligence & Data Science, for providing a very good facilities and all the support forwarded to us in carrying out this Project Work successfully.

We also like to thank our Project Coordinator **Mrs. K. Padma Priya, Assistant Professor,** Department of Artificial Intelligence & Data Science for her help and support provided to carry out the Project Work successfully.

Also, we are thankful to **Mrs. P S Geetha, Assistant Professor,** Department of Artificial Intelligence & Data Science for being our Project Guide, under whose guidance this project work has been carried out Project Work successfully.

We are also thankful to the teaching and non-teaching staff of Department of Artificial Intelligence & Data Science, KSSEM for helping us in completing the Project Work.

HARSHITHA V	**1KG22AD020**
M JOSHNA	**1KG22AD034**
MOKSHITHA S THOTA	**1KG22AD039**
R ALEKHYA	**1KG22AD045**

ABSTRACT

Urban intersections often experience severe congestion due to the limitations of traditional fixed-time traffic signals, which fail to account for real-time traffic flow. This project presents an AI-powered adaptive traffic signal control system that dynamically adjusts signal timings based on live vehicle density. The system employs a computer vision model for vehicle detection and traffic density estimation, and integrates it with SUMO (Simulation of Urban Mobility) to simulate an intersection and evaluate performance. Based on the detected traffic volume, the controller allocates optimal green-time durations to each lane, reducing waiting time and improving overall traffic flow. Experimental results from the SUMO simulation demonstrate that the proposed system significantly outperforms conventional static signal control by minimizing queue length, travel delay, and fuel wastage. This work highlights the potential of AI-driven traffic management systems in building smarter, more efficient urban transportation networks.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

Efficient traffic management has become one of the most pressing challenges in rapidly urbanizing cities, where increasing vehicle populations and complex road networks contribute to congestion, delays, and safety risks. Intersections, in particular, serve as critical control points in traffic flow and are often the primary sources of bottlenecks. Traditional traffic signal systems rely on predefined, static timers that operate without awareness of conditions. Such rigid systems struggle to accommodate fluctuating traffic volumes throughout the day, leading to unnecessary waiting times, wasted fuel, and decreased overall mobility. These inefficiencies highlight the urgent need for intelligent, adaptive solutions capable of responding dynamically to changing traffic patterns.

The development of automated, AI-driven traffic signal control systems has gained significant momentum due to advancements in computer vision and deep learning. Cameras deployed at intersections capture a continuous stream of visual information containing valuable cues about vehicle density, lane congestion, and movement patterns. By interpreting this data, machines can approximate the decision-making process of human traffic controllers adjusting green times, prioritizing busy lanes, and alleviating congestion in a more responsive manner.

The advent of deep learning has transformed this landscape by enabling highly accurate vehicle detection and traffic density estimation directly from raw camera footage. These models automatically extract salient spatial features, reducing dependency on handcrafted rules and improving robustness across varied environments. A critical component of modern traffic research involves validating these intelligent systems using realistic simulation platforms.

In this project, SUMO (Simulation of Urban Mobility) plays a central role in modelling an intersection environment, generating synthetic traffic flows, and analyzing the impact of AI-driven signal control algorithms. By integrating computer vision–based vehicle detection with SUMO's configurable simulation environment, the project demonstrates how adaptive traffic signal control can significantly enhance intersection performance. As research progresses, combining visual data with additional modalities such as GPS inputs, sensor networks, and historical traffic patterns promises to pave the way for fully intelligent, context-aware urban traffic systems that can streamline mobility and transform future smart cities.

1.1 Overview

Urban traffic congestion is a growing challenge in rapidly developing cities, where intersections often act as major bottlenecks in the transportation network. Traditional traffic signals operate on fixed-time cycles, offering the same green and red durations regardless of the actual traffic flow. This results in long waiting times, fuel wastage, increased pollution, and inefficient movement of vehicles. With advancements in artificial intelligence and computer vision, it is now possible to monitor traffic conditions using visual data and intelligently adjust signal timings.

This project focuses on the development of an AI-powered adaptive traffic signal control system that uses camera-based vehicle detection to measure traffic density and dynamically optimize signal timings. The system integrates a deep learning-based computer vision model for traffic analysis with SUMO (Simulation of Urban Mobility) to simulate realistic intersection scenarios. By combining detection with simulation-based evaluation, the project demonstrates how intelligent traffic control can significantly reduce delays, improve road efficiency, and enhance overall traffic management in urban environments.

1.2 Purpose of the Project

The purpose of this project is to develop an intelligent and adaptive traffic signal control system that addresses the limitations of conventional fixed-time traffic lights. By utilizing computer vision and deep learning techniques, the system aims to accurately detect vehicle density in and adjust signal timings dynamically to improve overall traffic flow. The project also seeks to reduce unnecessary waiting times, minimize congestion, and curb fuel consumption and pollution by preventing vehicles from idling at intersections. Through integration with the SUMO simulation environment, the effectiveness of the proposed system is evaluated under realistic traffic scenarios, demonstrating how AI-driven decision-making can enhance the efficiency, sustainability, and responsiveness of modern urban traffic management. Additionally, the project aims to showcase the potential of AI-based automation in solving everyday urban challenges, making traffic systems more intelligent and user-friendly.

1.3 Scope of the Project

The scope of this project encompasses the design, development, and simulation of an AI-driven adaptive traffic signal control system capable of responding to traffic conditions. The system focuses on detecting vehicles through computer vision techniques, estimating lane-wise traffic density, and dynamically adjusting signal timings based on the identified congestion levels. This project includes the implementation of a deep learning model for vehicle detection, integration of the detection output with a SUMO (Simulation of Urban Mobility) environment, and evaluation of the adaptive signal's performance under various simulated scenarios. The scope is limited to a single intersection simulation, and does not extend to hardware deployment or real-world installation. However, the system is designed in a scalable manner so that it can be expanded to multiple intersections or integrated with IoT-based traffic sensors in the future. The project also explores the potential benefits of intelligent traffic control in reducing congestion, improving mobility, and enhancing the efficiency of urban transportation systems.

1.4 Definitions

Adaptive Traffic Signal

Adaptive Traffic Signal systems represent an advanced form of traffic signal control in which the duration of green, red, and yellow lights is dynamically adjusted based on real-time traffic conditions instead of following a fixed-cycle timer. Traditional systems use a constant and repetitive schedule that does not consider actual traffic flow, resulting in long queues, unnecessary waiting, and increased fuel consumption. In contrast, adaptive systems collect real-time data from sensors, cameras, or simulation tools and make intelligent decisions about how long each direction should receive a green signal.

The core principle behind adaptive traffic control is demand-based optimization. If a particular approach at an intersection has more vehicles, the controller allocates a longer green time to that lane or phase. Conversely, lanes with fewer vehicles get shorter cycles, ensuring that the intersection operates efficiently. Adaptive systems continuously monitor parameters such as queue length, arrival rate, lane occupancy, and turning proportions. Based on these observations, they update signal timings at regular intervals sometimes as frequently as every few seconds.

Several algorithms are used in adaptive traffic systems, including rule-based logic, proportional allocation, fuzzy control, reinforcement learning, and AI-driven optimization. In modern smart cities, adaptive controllers help reduce congestion, minimize emission levels, improve travel time, and enhance the overall safety and smoothness of traffic flow. The ability to respond to accidents, sudden demand spikes, or uneven lane usage makes adaptive systems far superior to fixed-time signals. In summary, an Adaptive Traffic Signal is a dynamic, data-driven mechanism that intelligently manages traffic movements to improve road performance and user experience.

Computer Vision

Computer Vision is a specialized domain within Artificial Intelligence that focuses on enabling machines to see, interpret, and understand visual information from digital images or video streams. It allows computers to replicate human visual perception by recognizing patterns, detecting objects, analyzing scenes, and extracting meaningful insights from visual data. Computer Vision combines aspects of mathematics, image processing, machine learning, and neural network-based modeling to process pixels and transform them into actionable information.

At its core, Computer Vision involves techniques such as filtering, edge detection, segmentation, feature extraction, classification, and tracking. Modern approaches rely heavily on deep learning, particularly Convolutional Neural Networks (CNNs), which learn hierarchical visual features automatically. These models are trained using large datasets and can then recognize objects like vehicles, pedestrians, traffic signs, faces, or obstacles with high accuracy.

In the context of intelligent transportation systems, Computer Vision plays a crucial role in tasks such as vehicle detection, traffic density estimation, license plate recognition, and real-time monitoring. Cameras placed at intersections or road segments serve as sensors that continuously capture traffic flows. Computer Vision algorithms analyze these feeds to determine congestion levels, detect violations, or support adaptive signal operations. Its ability to operate autonomously, process massive amounts of data, and deliver accurate real-time results makes it essential for developing smart traffic systems and modern AI-driven transportation applications.

Vehicle Detection

Vehicle Detection refers to the automated process of locating and identifying vehicles such as cars, trucks, buses, motorcycles, and rickshaws in an image or video frame. It is a key application of Computer Vision, especially within traffic surveillance, intelligent transportation systems, and autonomous driving. The objective of vehicle detection is not only to recognize the presence of vehicles but also to determine their exact positions within the scene, typically represented through bounding boxes.

Traditional vehicle detection techniques used handcrafted features such as Haar cascades, HOG (Histogram of Oriented Gradients), and background subtraction. Although these methods worked in controlled environments, they often failed under real-world conditions with varying lighting, occlusion, shadows, or complex backgrounds. Modern vehicle detection systems use deep learning models particularly YOLO (You Only Look Once), SSD (Single Shot Detector), and Faster R-CNN which offer far greater accuracy and speed. YOLOv8, used in this project, processes images in a single forward pass, making it capable of real-time detection at high frame rates.

Vehicle detection provides vital inputs to adaptive traffic systems. By counting the number of vehicles in each lane, the system determines traffic density and allocates green time accordingly. It also helps identify peak times, detect illegal movements, and support tasks such as traffic analysis, congestion prediction, and autonomous navigation. In summary, vehicle detection transforms raw visual data into structured, meaningful information that supports intelligent decision-making in traffic management.

SUMO (Simulation of Urban Mobility)

SUMO (Simulation of Urban Mobility) is an open-source, microscopic traffic simulation framework widely used in research, transportation planning, and intelligent mobility systems. SUMO allows users to build realistic traffic networks, define vehicle routes, simulate intersection operations, and analyze traffic performance under different scenarios. It is highly flexible, supporting custom road designs, adaptive signal controls, detectors, public transport modeling, and multimodal mobility dynamics.

SUMO's microscopic approach means that each vehicle is simulated individually, with its own speed, acceleration, lane-changing behavior, and route selection. This level of detail enables accurate modeling of urban intersections, expressways, roundabouts, and complex traffic environments. SUMO includes a suite of tools such as NetConvert (for building networks),

DuaRouter (for generating routes), SUMO-GUI (for visualization), and TraCI (Traffic Control Interface), a powerful API that allows external Python programs to interact with the simulation in real time.

In this project, SUMO serves as the testing environment for evaluating both fixed-time and adaptive, AI-powered signal control systems. The adaptive controller uses TraCI to retrieve lane vehicle counts and update signal phases dynamically based on YOLO-based detection or SUMO's internal metrics. SUMO provides detailed performance outputs such as waiting times, queue lengths, travel times, and throughput, enabling scientific comparison between different traffic-control strategies.

Because SUMO is open-source, it is widely adopted in academic and industry research, making it ideal for experimentation, algorithm testing, and intelligent transportation development. In summary, SUMO forms the foundation of the simulation environment in this project, enabling accurate, controlled, and repeatable evaluation of adaptive traffic signal strategies.

Traffic Density

Traffic Density is a key traffic engineering parameter that quantifies the number of vehicles present on a road segment or at an intersection at a given moment. It indicates how congested a lane or junction is and plays a critical role in analyzing road performance, predicting traffic flow, and designing control strategies. Density is often measured as vehicles per lane or vehicles per kilometer, but in real-time systems, it is commonly represented as the count of vehicles observed in a camera frame or through sensors.

In adaptive traffic signal systems, density acts as the primary input for determining signal timings. A high density indicates heavy congestion, requiring longer green time to clear queues, while low density suggests that shorter green durations are sufficient. Real-time density estimation allows a controller to dynamically adjust to traffic fluctuations, improving throughput and reducing total waiting time.

Density can be computed through several methods:

1. Computer Vision-based density, where deep learning models detect and count vehicles.
2. Inductive loop sensors, installed under the road surface.
3. Radar/LiDAR sensors, used in high-end traffic systems.
4. Simulation-based density, from SUMO's internal lane counters.

Traffic density is also relevant in research on traffic flow theory, where it forms part of the fundamental relationships between speed, flow, and congestion. By continuously monitoring density, adaptive systems can prevent excessive queue buildup, reduce stop–go waves, and optimize the movement of vehicles at intersections. In summary, traffic density represents the "pulse" of the road system and is essential for intelligent, data-driven traffic control.

Vehicle Annotation

Annotation is the process of manually or semi-automatically labeling important objects, regions, or features within images and videos to create structured data required for training computer vision models. In simple terms, annotation transforms raw visual content into meaningful information that a machine can learn from. In the context of AI-based traffic management, annotation typically involves marking vehicles such as cars, buses, trucks, two-wheelers, and autorickshaws by drawing bounding boxes or segmentation masks on each object. This labelled data serves as the ground truth that a deep learning model uses to understand what a vehicle looks like, how it appears from different angles, and how it behaves in various lighting or traffic conditions.

Annotation is a crucial step in building reliable AI systems because machine learning models learn patterns directly from the annotated dataset. The accuracy, consistency, and completeness of the annotations heavily influence the performance of the final model. Poor-quality annotation can lead to incorrect predictions, misclassification of vehicles, and unstable real-time inference. Therefore, high-quality annotation requires careful attention to object boundaries, proper labelling of classes, removal of ambiguous frames, and maintaining uniform labelling standards across the entire dataset.

Modern annotation tools such as CVAT, Label Studio, Roboflow, and LabelImg simplify this process by providing user-friendly interfaces for drawing bounding boxes and exporting datasets in YOLO format. Some tools also support auto-annotation using AI assistance, which speeds up the labelling process for large datasets. For traffic applications, annotation may also involve labelling lanes, traffic lights, occluded vehicles, or multiple vehicle categories to allow the detection model to distinguish between different road users.

In this project, annotation plays a foundational role in training the YOLOv8 vehicle detection model. Thousands of traffic images collected from different sources are annotated to ensure the model learns to recognize diverse vehicle types under varying conditions such as day/night lighting, shadows, rain, or congestion. The annotations are saved in YOLO format, where each label file contains the class name and the normalized coordinates of the bounding box. Once

annotation is complete, the dataset is split into training, validation, and testing sets, ensuring the model learns effectively and generalizes well to unseen images.

In summary, annotation is the essential preparation step that converts unstructured visual data into machine-readable labels. It bridges the gap between raw images and AI learning, enabling deep learning models to detect vehicles accurately and support intelligent adaptive traffic signal systems. Without proper annotation, the AI model would not understand what a vehicle looks like, making accurate traffic density estimation impossible.

YOLOv8n

YOLOv8n (You Only Look Once Version 8 – Nano variant) is the smallest and fastest model in the YOLOv8 family developed by Ultralytics for real-time object detection, classification, and segmentation tasks. The "n" in YOLOv8n stands for nano, indicating that it is optimized for high speed and low computational cost while maintaining competitive accuracy. YOLOv8n is built using a modern deep-learning architecture that combines convolutional layers, C2f modules, and efficient feature aggregation to detect objects in a single forward pass. Its lightweight design makes it exceptionally well-suited for real-time applications such as traffic monitoring, surveillance, mobile deployment, embedded systems, and edge computing devices with limited processing power.

Despite its small size, YOLOv8n delivers strong performance due to its innovative architectural components. It includes a backbone for feature extraction, a neck for multi-scale feature fusion, and a head for predicting bounding boxes, class probabilities, and object confidence scores. YOLOv8n's one-stage detection design means the model directly predicts object locations and classes without requiring region proposal steps, making it extremely efficient. Its anchor-free mechanism simplifies the detection process and improves generalization across diverse object sizes and shapes an important advantage when processing complex scenes like road intersections with varying vehicle types.One of the major strengths of YOLOv8n is its ability to operate at high frame rates, often exceeding 100 FPS on modern GPUs and running smoothly even on low-power devices. This real-time capability is essential in adaptive traffic signal systems, where fast and accurate vehicle detection is necessary for making time-sensitive decisions about signal timing. In this project, YOLOv8n is used to detect vehicles such as cars, buses, bikes, trucks, and autorickshaws from camera frames or traffic images, providing accurate vehicle counts for traffic density estimation. These counts are then used by the adaptive signal control algorithm to allocate optimal green time for each lane.

YOLOv8n also integrates seamlessly with popular annotation formats like YOLO, COCO, and Pascal VOC, allowing easy training on custom datasets labelled using tools like CVAT or Roboflow. The training process is efficient due to the model's small parameter size, enabling faster convergence and lower training GPU requirements. Additionally, YOLOv8n supports extensive data augmentation, improved loss functions, and advanced training strategies that enhance accuracy even with modest datasets. In summary, YOLOv8n is a highly efficient, lightweight object detection model that plays a crucial role in this project by enabling accurate vehicle detection at high speeds.

TraCI (Traffic Control Interface)

TraCI, short for Traffic Control Interface, is a powerful API that enables real-time, bidirectional communication between the SUMO (Simulation of Urban Mobility) traffic simulator and external programs such as Python, C++, Java, or MATLAB applications. It acts as a bridge that allows developers to control, monitor, and manipulate a SUMO simulation while it is running. Through TraCI, external scripts can retrieve live traffic data such as lane vehicle counts, queue lengths, speeds, or signal states and modify the simulation environment by changing traffic light phases, rerouting vehicles, adjusting speeds, or injecting new vehicles dynamically. This capability makes TraCI essential for creating intelligent, adaptive traffic management systems.

TraCI operates on a client–server model, where SUMO runs as the server and listens on a specified port (remote-port), while an external Python script acts as the client. Once the connection is established, the client can issue commands at every simulation step. This real-time stepping mechanism allows algorithms such as AI controllers, reinforcement learning agents, or rule-based adaptive controllers to make decisions based on the latest observed traffic conditions. In this project, TraCI is used to fetch lane-wise vehicle counts and update green times adaptively, making it the central component that connects the YOLO-based detection logic with the SUMO environment.

One of TraCI's most important features is its extensive set of modules, including traci.lane, traci.vehicle, traci.edge, and traci.trafficlight. These modules provide granular access to nearly every element of the simulation. For instance, traci.trafficlight.setPhaseDuration() allows updating a signal phase on the fly, while traci.lane.getLastStepVehicleNumber() retrieves the number of vehicles currently waiting in a lane. The ability to control traffic lights in real time enables implementation of advanced adaptive systems that outperform fixed-time controllers.

TraCI is also widely used for research in connected vehicles, smart mobility, emergency routing, congestion management, and autonomous driving. Because it allows full programmability from an external script, developers can integrate machine learning models, optimization algorithms, reinforcement learning agents, and computer vision models into traffic simulations. This flexibility makes TraCI one of the key technologies in modern Intelligent Transportation Systems (ITS).

In this adaptive traffic signal project, TraCI plays a foundational role by enabling the Python controller to speak directly to SUMO. It allows the system to continuously measure traffic density, calculate optimal green times, and update signal states every few seconds. Without TraCI, it would not be possible to implement real-time, AI-driven signal control within SUMO.

1.5 Report Organization

This report is organized into a series of well-structured chapters to ensure clarity, logical flow, and comprehensive understanding of the AI-Powered Traffic Signal Processing System.

Chapter 1 begins with an introduction to the problem of urban traffic congestion, followed by the project's purpose, scope, and key definitions necessary to understand the system.

Chapter 2 presents a detailed literature survey, summarizing 40 research papers that highlight existing approaches, technologies, and research gaps in intelligent traffic signal control. **Chapter 3** outlines the problem identification, problem statement, goals, and objectives of the project, clearly defining the motivation and direction of the proposed solution.

Chapter 4 focuses on the system design, describing the architecture, data flow diagrams, requirement analysis, and the objective-wise design of each system component.

Chapter 5 explains the implementation process, including dataset preparation, YOLOv8 model training, adaptive signal logic development, and SUMO-TraCI integration.

Chapter 6 presents the results, performance evaluation, comparisons between fixed-time and adaptive systems, and visual outputs from the simulation.

Chapter 7 concludes the report with major findings, limitations, and potential future enhancements, followed by references and appendices for supporting materials. This structured organization ensures smooth navigation across all stages of the project from ideation to system execution.

Chapter 2

LITERATURE SURVEY

The literature survey conducted in Phase I aimed to understand existing research and techniques in deepfake detection. A total of 40 research papers were reviewed and this survey helped identify research gaps, compare algorithms, and build a solid foundation for developing an efficient Traffic signal processing system in later project phases.

Table 2.1 Literature Survey Summary

SL.NO	AUTHORS	TITLE OF THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
1	Xuanning Zhang	Artificial Intelligence in Intelligent Traffic Signal Control System	Deep Q-Network (Reinforcement Learning)	Improves traffic flow, reduces travel time, congestion, and accidents.
2	Syed S.S.M. Qadri, Mahmut A Gokce,Erdinc Oner	State-of-the-Art Review of Traffic Signal Control Methods	AI, Metaheuristics, Microsimulation	AI improves optimization; most studies use simulation-based evaluation.
3	Taoyu Pan	Traffic Signal Control via Reinforcement Learning	Deep Q-Learning	Reduced waiting time up to 100%, reduced queue length and travel time.
4	Roopa Ravish, Shanta Ranga Swamy	Smart Traffic Control: Answer Overview of Issues, Solutions & Future Directions	AI, ML, CNN, LSTM	AI-based models enhance traffic prediction and congestion control.
5	Selim Reza, Hugo S. Oliveira, Jose J.M.Machado, Joao M.R.S. Tavares	Urban Security: An Image Processing & Deep-Learning-Based Smart Traffic Control and Regulation System	CNN, RNN, LSTM, GAN	Deep learning achieves >91% accuracy in traffic control and prediction.

SL.NO	AUTHORS	TITLE OF THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
6	Nishant Kumar, Martin Raubal	Applications of Deep Learning for Detection, Prediction, and Relief of congestion: A Survey	CNN, LSTM, GRU, DCRNN	DL models outperform traditional methods; accuracy 84–96%.
7	SooJean Han, Soon-Jo Chung, Johanna Gustafson	Structural & Temporal Pattern Learning for Vehicle Traffic Network Congestion Control	PLMP (Pattern-Learning Memory Model)	Reduced waiting time by 97% and improved signal efficiency.
8	BibekShrestha Vishal Singh, Gaurav Darlami,Cabin Upadhyay	AI-Based Traffic Management System	YOLOv8 + Arduino	detection enabled reducing signal cycle from 140s to 60s.
9	Muhammad Saleem, Sagheer Abbas, Taher M. Ghazal, Muhammad Adnan Khan, Nizar Sahawneh	Smart-City: Fusion-Based Smart Traffic Congestion Control System for Vehicular via Machine Learning	ANN, SVM, Fuzzy Logic, IoV	Fusion model achieved 95% congestion prediction accuracy.
10	Hameed Khan, Jitendra Singh	Smart Traffic Control:Using Machine Learning for Dynamic Road Traffic Control in Urban	YOLOv3, CNN, OpenCV	88% accuracy in detection; improved adaptive traffic signal timing.

SL.NO	AUTHORS	TITLE OF THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
11	Yousra S. Almathami	A Proposal to a Dynamic Traffic Detection System in Saudi Arabia: A Sun-Powered Drones Approach	YOLOv8, CCTV + Solar-powered drones	Hybrid drone CCTV system improves detection, activates drones during congestion.
12	M. Pavan Kumar Reddy et al.	AI Driven Traffic Management System Using Computer Vision and ML (2021)	YOLO, SSD, RNN, LSTM, Reinforcement Learning	Improved traffic monitoring, adaptive control, and reduced congestion.
13	Ruhul Amin Khalil et al.	Advanced Learning Technologies for Intelligent Transportation Systems (2024)	CNN, RNN, GAN, GCN, Autoencoders, Deep RL	DL improves prediction, detection, classification; challenges include standardization.
14	Auwal A. Musa et al.	Sustainable Traffic Management for Smart Cities Using IoT-Oriented ITS (2023)	AI sensors, ML, IoT, Cloud computing	Better traffic prediction, pollution reduction, hybrid CAV–HDV integration.
15	Ayad G. Ismaeel et al.	Enhancing Traffic Intelligence Using Sustainable Deep Radial Function (2023)	Deep RBF Neural Network	Higher accuracy in congestion detection & forecasting vs traditional models.
16	Ali Rizwan et al.	Simulation of IoT-Based VANETs for Smart Traffic Management (2022)	NFV, DDP4V protocol, IoT simulations	DDP4V achieves faster packet delivery & coverage in high-density traffic scenarios.
17	Mamoona Humayun et al.	Smart Traffic Management System for Metropolitan Cities Using Cutting Edge Technologies (2022)	IoT sensors, 5G, Cloud computing, Google Maps	Reduced congestion through dashboards, better routing.

SL.NO	AUTHORS	TITLE OF THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
18	Karol Jurczenia, Jacek Rak	Survey of Vehicular Network Systems for Road Traffic Management (2022)	V2V, V2I, V2X; Cloud, Fog, MEC architectures	Low-latency MEC & Fog computing essential for ITS deployment.
19	Anakhi Hazarika et al.	Edge ML Technique for Smart Traffic Management in ITS (2024)	YOLO, Raspberry Pi (Edge ML), LoRaWAN, IEEE 802.15.4 DSME	Reduced waiting time, emergency vehicle priority, improved traffic flow.
20	S. B. Raheem et al.	The Cause, Effect and Possible Solution to Traffic Congestion on Nigeria Road (2015)	Manual traffic counting, LOS analysis, density/volume study	Peak congestion, delays, poor road conditions, and lack of infrastructure identified.
21	Mahima Jaiswal, Neetu Gupta, Ajay Rana	Traffic Management in Emergency using Artificial Intelligence (2020)	Variable signal timing, unmanned vehicle routing, reservation-based system, predictive traffic control	Tracks ambulance in real time; reduces congestion using intelligent traffic signal management.
22	C. Heltin Genitha, S. Abishek Danny, A.S. Hepsi Ajibah	AI-based real time Traffic Signal Control System using Machine Learning (2023)	YOLO object detection, real time density calculation, adaptive signal switching	Reduces congestion, delays, fuel consumption, and improves traffic flow.
23	Monica C, Bangalore Jyothi, Amogh Ramagiri	Intelligent Traffic Monitoring, Prioritizing and Controlling Model based on GPS (2023)	GPS, IoT sensors, AI + Deep Learning prediction	Improves traffic flow and safety using real time GPS + IoT predictions.

SL.NO	AUTHORS	TITLE F THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
24	Hanaa Abohashima, Mohamed Gheith, Amr Eltawil	IoT-based Smart Traffic Lights Control System within a V2X Framework (2020)	V2V, V2I, V2X communication, IoT, AI, neuro-fuzzy control	More adaptive than fixed controllers; dynamic signal adjustment improves traffic movement.
25	Ke Gu, Jieyu Hu, Weijia Jia	Adaptive Area-Based Traffic Congestion Control & Management using Fog Computing (2023)	Fog computing IoV, regional traffic correlation analysis	Guides traffic flow in real time; alleviates congestion using fog-level distributed processing.
26	Md. Imran Uddin, Md. Shahriar Alamgir, Md. Mahabubur Rahman	AI Traffic Control System Based on Deepstream and IoT Using NVIDIA Jetson Nano (2023)	Edge AI, Deepstream, Jetson Nano, IoT cameras, number plate detection	Detects rule violations, identifies number plates, sends automated penalty notifications.
27	Jiawei Li, Dong Xie, Qiyi Zhu, Zhishang Wu	Construction of Intelligent Transportation Information Management System (2023)	AI-based traffic volume prediction, centralized traffic hub	Reduces delay time by 50%; strong real-time monitoring and decision support.
28	Mohammed Yasin Arafat, Mirza Fuad Adnan	AI-based Affordable High-density Traffic Monitoring System (2023)	YOLOv3, traffic density estimation, low-cost edge implementation	Autonomous monitoring reduces accidents and improves road navigation safety.
29	Ahgalya Subbiah	AI-Enhanced IoT System for Efficient Traffic Management (2024)	IoT speed cameras, CNN feature extraction, SVM classification, cloud analytics	High accuracy and fast processing improves congestion prediction in smart cities.

SL.NO	AUTHORS	TITLE OF THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
30	S. Sirphy, S. Thanga Revathi	Adaptive Traffic Control System Using YOLO (2023)	YOLO vehicle detection + density-based signal switching algorithm	Adjusts green signal time dynamically based on real-time traffic density.
31	N. Kumar, M. Alazab, A. A. Abd El-Latif	Overview of Road Traffic Management Solutions Based on IoT and AI (2021)	IoT-based systems, Q-learning, CSA–VIKOR, ARIMA	AI + IoT improve routing and traffic signal control; qualitative insights; no numerical evaluation.
32	Luca Fredianelli et al.	Traffic Flow Detection Using Camera Images and ML Methods in ITS (2022)	YOLOv2, Low-cost CCTV integration, Noise mapping system	mAP = 92%; integrates traffic + environmental monitoring; sensitive to lighting/weather.
33	Karan Singh, Nikita Malik	CNN-Based Approach for Traffic Sign Recognition System (2021)	CNN, Spatial Transformer Layer	Accuracy: 95% train, 90.3% test; dataset limitations affect generalization.
34	Maryam Shaygan et al.	Traffic Prediction Using Artificial Intelligence (2022)	Deep Learning, Federated Learning, Blockchain-based ITS	Discusses metrics (MAE, RMSE, R^2). Summarizes literature; no numerical experiments.
35	Sayed A. Sayed, Yasser A. Hamid, Hesham A. Hefny	Artificial Intelligence-Based Traffic Flow Prediction (2023)	RF, SVM, CNN, LSTM, SG-CNN	SGR R^2=0.90; SG-CNN MAE=0.01578; challenges: adaptability.
36	Jamal M. Assbeihat, Nosheen Rafi	Management of Artificial Intelligence Traffic Systems in Smart Cities (2021)	RFID, GPS, CCTV-based VIP data exchange	AI enables smart mobility; lacks performance metrics; standardization issues.

SL.NO	AUTHORS	TITLE OF THE PAPER	ALGORITHMS / TECHNOLOGY USED	KEY FINDINGS FROM THE PAPER
37	Rama Chandra Rao Nampalli	Leveraging AI in Urban Traffic Management (2021)	Centralized & distributed AI architectures	AI reduces congestion; scalability and interoperability issues.
38	Vishal Mandal, Abdul R. Mussah	Artificial Intelligence-Enabled Traffic Monitoring System (2020)	Mask R-CNN, YOLO, Faster R-CNN, Anomaly detection	Queue detection: Precision 92.8%; YOLO accuracy ~93.7%; suffers in low-light.
39	F. A. Al-Habashna, M. Al-Rousan, A. M. Rousan	Efficient Dynamic Traffic Light Scheduling Considering Emergency Vehicles (2020)	MAX-GREEN / BEST-GREEN logic	Reduces waiting time; simulation only, no prototype.
40	N. Tarchi, F. Granelli, A. Capone, et al.	Internet of Smart-Cameras for Traffic Lights Optimization in Smart Cities (2021)	Fuzzy logic, Distributed smart camera network, CV	Reduces queue time; ~3.8 FPS; hardware-dependent, not tested in harsh weather.

Chapter 3

PROBLEM IDENTIFICATION

Managing traffic flow in rapidly growing urban areas has become a major challenge due to increasing vehicle density and unpredictable traffic patterns. Traditional traffic signal systems rely on fixed timers that cannot adapt to real-time road conditions, leading to unnecessary delays, congestion, and fuel wastage. During peak hours, some lanes become heavily crowded while others remain underutilized, highlighting the inefficiency of static signal control. Manual monitoring of traffic is also time-consuming and prone to human error, making it unsuitable for dynamic environments. As a result, emergency vehicles often face delays, contributing to safety risks and slower response times. Additionally, existing systems lack intelligent decision-making capabilities to optimize vehicle movement. These problems collectively reduce road efficiency and increase pollution levels. To address these issues, an automated and adaptive solution is required. An AI-based traffic signal system that detects vehicles in real time can intelligently adjust green-light duration. This approach can significantly improve traffic flow, reduce congestion, and enhance overall transportation efficiency.

3.1 Problem statement

Urban traffic congestion continues to be one of the most persistent challenges in modern cities, leading to increased travel time, fuel consumption, carbon emissions, and overall commuter frustration. Traditional traffic signal systems operate using fixed-time cycles or simple sensor-based triggers, which fail to adapt to rapidly changing and highly dynamic traffic conditions. As a result, these systems cannot efficiently handle peak-hour congestion, sudden traffic surges, or emergency vehicle prioritization. The core problem addressed by this project is the absence of an intelligent, adaptive, and data-driven traffic signal management system capable of decision-making. Existing traffic control mechanisms lack situational awareness, rely heavily on manual tuning, and do not incorporate computer vision analytics to understand vehicle density, queue length, or lane-wise movement patterns. Furthermore, current solutions do not integrate predictive models to anticipate congestion nor provide lane prioritization during critical events such as ambulance movement or bottleneck formation. The lack of such an automated AI-based system results in inefficient traffic flow, increased waiting time at intersections, and delayed emergency responses. Hence, there is a clear and urgent need for a next-generation platform that leverages video data, computer vision algorithms, and adaptive signal optimization to intelligently manage traffic flow, reduce congestion.

3.2 Project Goals

The project goal is defined by the development and deployment of AI Powered Traffic Signal Processing, an intelligent, next-generation platform designed to dynamically optimize traffic signal timing using live video input and simulation-based evaluation. This project focuses on building a robust system that integrates deep learning–based vehicle detection, traffic density estimation, and adaptive signal switching to overcome the limitations of traditional fixed-timing systems. The core deliverable is a fully functional Python-based application capable of processing traffic video streams, detecting vehicle density using YOLOv8, and adjusting signal durations accordingly, with all evaluations performed within the SUMO traffic simulation environment. The analytical and functional scope is intentionally broad, covering multiple AI-driven components essential for smart traffic management.

The system is scoped to perform:

- **Vehicle Detection & Tracking**, using a YOLOv8 model trained on annotated traffic datasets, combined with OpenCV-based vehicle counting and lane-wise density estimation for accurate analysis.
- **Adaptive Signal Control**, where the system dynamically computes the optimal green-time duration based on vehicle density, queue length, and lane priority rules, enabling intelligent and responsive traffic flow management.
- **Traffic Simulation & Evaluation**, using SUMO to test, validate, and visualize adaptive traffic control, measure improvements in waiting time, vehicle throughput, and congestion reduction, and compare adaptive signals against traditional fixed cycles.

Functionally, the scope begins with collecting and annotating raw traffic videos, proceeds through model training, detection, and integration with SUMO, and concludes with system performance evaluation and visual dashboards illustrating results. The project is limited to software-based simulation and does not involve real-world hardware deployment or integration with existing municipal traffic systems. It is intended solely for academic, research, and demonstration purposes, focusing on video-based AI analytics without incorporating IoT sensors, GPS data, or multi-intersection coordination.

3.3 Project Objectives

Objective 1: Dataset Collection & Annotation

To collect, annotate, and structure traffic videos for training deep learning models capable of identifying and tracking vehicles in various traffic scenarios.

Objective 2: Model Training for Vehicle Detection

To train and validate YOLOv8-based detection models using annotated datasets to accurately detect different vehicle types and estimate lane-wise traffic density.

Objective 3: Adaptive Signal Switching

To implement an intelligent traffic signal control algorithm that dynamically adjusts signal durations based on vehicle density and lane congestion levels.

Objective 4: Traffic Simulation & System Evaluation

To simulate traffic scenarios using SUMO and evaluate the performance of the adaptive traffic signal system by comparing improvements in waiting time, queue length, and traffic flow efficiency.

Chapter 4

SYSTEM DESIGN

System design is a crucial step in developing an AI-based traffic signal control system because it defines how all components of the project work together to solve the problem of traffic congestion. It provides a structured blueprint that explains how data is collected, how the AI model processes this information, and how the system makes real-time decisions to adjust signal timings. Through system design, we break the project into modules such as data input (simulated cameras), vehicle detection, traffic analysis, and signal optimization, ensuring each part functions smoothly and communicates effectively with the others. A well-designed system helps us build a reliable, scalable, and efficient traffic management solution. It also ensures that the AI model can adapt to varying traffic conditions and deliver accurate results. Overall, system design forms the foundation for implementing and testing the entire intelligent traffic signal project.

4.1 System Architecture

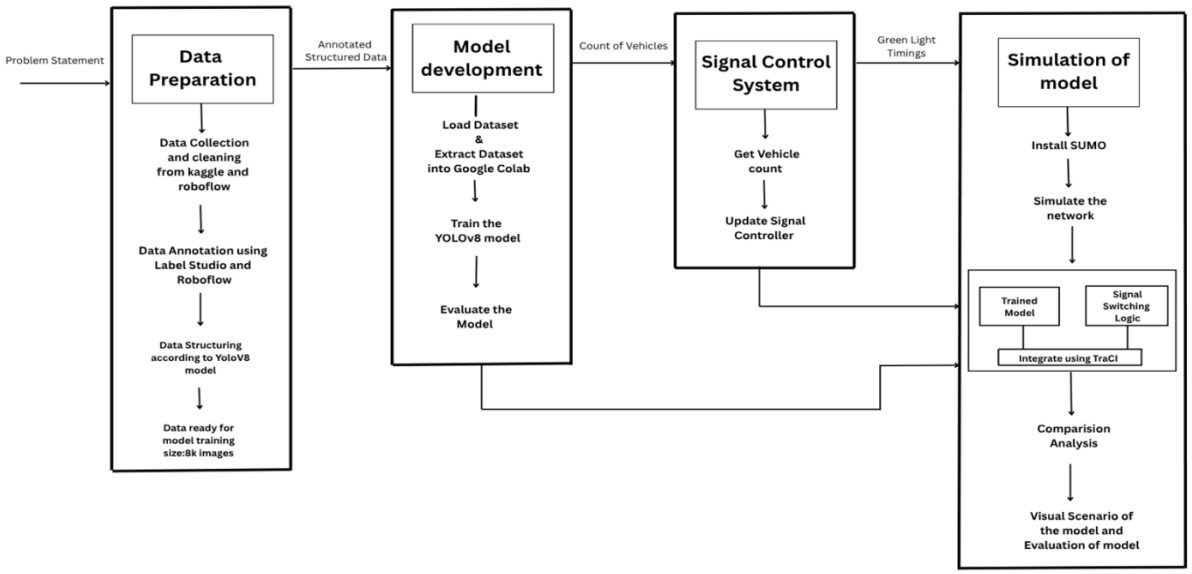


Fig. 4.1 System Architecture

The system architecture is designed as a structured pipeline that seamlessly integrates data processing, deep learning, real-time decision-making, and traffic simulation to build a fully functional AI-powered adaptive traffic signal control system. It is divided into four core modules Data Preparation, Model Development, Signal Control System, and Simulation of the Model each responsible for a specific stage in the development lifecycle. The architecture begins with large-scale dataset collection, annotation, and formatting to ensure high-quality

input for the model.

This curated data is then used to train a YOLO-based vehicle detection model capable of accurately identifying and counting vehicles under varying traffic conditions. The detected vehicle count becomes the foundation for the Signal Control System, where adaptive logic dynamically computes optimal green-light durations based on real-time density patterns. Finally, the entire system is integrated and evaluated within the SUMO simulation environment using TraCI, enabling a realistic assessment of performance improvements such as reduced waiting time, minimized congestion, and smoother traffic flow. This modular architecture not only supports scalability and flexibility but also ensures that every component from vision model to signal timing is validated through systematic testing and simulation.

Data Collection:

This step focuses on gathering a large and diverse set of traffic images and videos.

- **Identify Data Sources:**
  Select publicly available datasets from platforms such as **Kaggle, Roboflow Universe**, UA-DETRAC, or COCO.
- **Download Images/Videos:**
  Collect images showing cars, bikes, buses, trucks, and mixed traffic under different lighting and environmental conditions.

Data Cleaning:

Cleaning ensures that only high-quality visuals are used for model training.

- **Remove Blurry or Damaged Images:**
  Discard images affected by motion blur, low resolution, or improper lighting.
- **Filter Out Irrelevant Images:**
  Remove images with no vehicles or incorrect scenes unrelated to traffic.
- **Standardize Image Dimensions:**
  Resize extremely large images for uniformity and reduced processing load.
- **Frame Extraction (for videos):**
  Convert videos into individual frames using OpenCV.

Data Annotation:

Annotation is the most critical step in supervised learning for object detection.

- **Import Images into Annotation Tool:**
  Upload cleaned images to Label Studio, Roboflow Annotate, or CVAT.
- **Draw Bounding Boxes:**
  Manually mark vehicles in each image using annotation tools.
- **Review and Validate Annotations:**
  Double-check for incorrect, missing, or duplicate bounding boxes.

Dataset Structuring

This step prepares the dataset in the correct format for YOLOv8 training.

- Separate Dataset into Splits:
  Divide annotated data into:
  Train (70%), Validation (20%), Test (10%).
- Convert Annotations to YOLO Format:
  Convert each bounding box into YOLO's text format:
  class_id x_center y_center width height (normalized)
- Organize Folder Structure:
  Standard YOLOv8 structure:

```
dataset/
    images/
        train/
        val/
        test/
    labels/
        train/
        val/
        test/
```

Fig. 4.2 Yolov8 Folder Structure

Dataset Loading and Environment Setup:

- **Upload Dataset to Google Colab / Local Machine:**
  The structured YOLOv8 dataset is uploaded to Colab for GPU-based training.

- **Mount Google Drive (If using Colab):**

  Used for saving model weights, logs, and training outputs.
- **Install Required Libraries:**

  Install YOLOv8, PyTorch, OpenCV, and other dependencies using:
  - pip install ultralytics
- **Verify GPU Availability:**

  Check Colab's GPU (T4/P100/V100) to ensure faster training.

Outcome:

A fully configured training environment ready for model development.

Extract and Organize Dataset:

- **Unzip Dataset:**

  Extract the dataset folder into the Colab workspace.
- **Check File Structure:**

  Confirm YOLOv8 folder readiness.
- **Validate Annotation Files:**

  Ensure each image has a corresponding .txt label file.
- **Load dataset.yaml:**

  This YAML file includes class names, paths, and training configuration.

Outcome:

Dataset is verified, formatted correctly, and ready for training.

Training the YOLOv8 Model:

- **Select YOLOv8 Variant:**

  Choose YOLOv8n / YOLOv8s / YOLOv8m depending on speed vs accuracy.
- **Initialize Training Command:**

  Example:

yolo detect train data=dataset.yaml model=yolov8s.pt epochs=50 imgsz=640

- **Monitor Training Metrics:**
  Track:
  - Training loss
  - Precision
  - Recall
  - mAP (mean Average Precision)
- **Adjust Hyperparameters (if needed):**
  Tune learning rate, epochs, batch size, or image size for improved performance.

Outcome:

A trained YOLOv8 model with learned weights for detecting vehicles.

Signal Control System

- **Function:**
  This module uses the output of the detection model to dynamically adjust traffic signal timings.
- **Process:**
  The trained YOLOv8 model receives video frames and identifies the vehicle count per lane.
  Based on density, the module computes optimal green, yellow, and red timings.
  The adaptive signal logic replaces traditional static timers by updating the signal cycle according to real-time traffic demand.
- **Output:**
  Optimized green-light timings that are sent to the SUMO simulation environment.

Simulation of the Model

- **Function:**
  To evaluate the performance of the adaptive traffic signal system using a realistic traffic simulation.
- **Process:**
  SUMO is installed and a traffic network (intersection) is created using XML Files.

The network is simulated using real or synthetic traffic flows. Python–TraCI interface is used to integrate the YOLOv8 detection model with SUMO. The signal switching logic updates the SUMO-controlled traffic lights in real time. Comparative analysis is performed between fixed-time traffic signal and AI-based adaptive traffic signal

- **Output:**

  Visual representation of how adaptive control improves traffic flow.

4.2 Objective-wise Design

Objective 1: Dataset Collection & Annotation

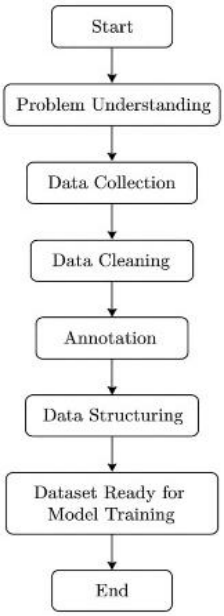


Fig. 4.3 Data Flow Diagram of Objective 1

The data flow diagram illustrates the complete workflow involved in preparing the dataset for the traffic detection model. The process begins with the Start phase, followed by a crucial step called Problem Understanding, where the requirements of the traffic detection task are clearly analyzed. Once the objective is defined, the next step is Data Collection, during which traffic images and videos are gathered from reliable sources such as Kaggle, Roboflow, or manually captured footage. After collecting the raw data, the workflow proceeds to Data Cleaning, where irrelevant, blurry, or duplicate images are removed to ensure high quality. The cleaned data is then passed to the Annotation stage, where tools like CVAT, Label Studio, or Roboflow Annotate are used to mark and label vehicles with bounding boxes. These annotations help the model recognize different vehicle categories. The next phase, Data Structuring, organizes the annotated images into a structured format required by the YOLOv8 model, including

separate folders for training, validation, and testing, along with YOLO-formatted label files. When all steps are completed, the dataset becomes Ready for Model Training, ensuring it is clean, annotated, and structured properly. Finally, the process reaches the End, indicating successful preparation of a high-quality dataset ready for deep learning model development.

Objective 2: Model Training for Vehicle Detection

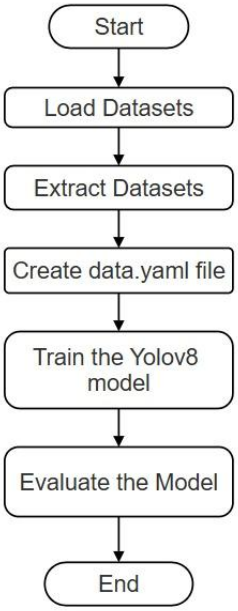


Fig. 4.4 Data Flow diagram for Objective 2

The Model Development Data Flow Diagram illustrates the complete workflow involved in training the YOLOv8 deep learning model for vehicle detection in the adaptive traffic signal control system. The process begins at the Start node and progresses to the first major task, Load Datasets, where the prepared dataset containing annotated images and YOLO-formatted labels is imported into the training environment such as Google Colab or a local machine. This step ensures that all necessary files, including images and label directories, are available and accessible for further processing. Once the dataset is loaded, the next step is Extract Datasets, which involves unzipping the dataset, verifying file structures, and organizing the images into training, validation, and testing folders. During this stage, the model developer checks whether every image has a corresponding annotation file and whether the dataset meets the requirements for YOLOv8 training. This extraction step is crucial because improper dataset arrangement can lead to model errors or incorrect training paths.

Following extraction, the workflow proceeds to the creation of the data.yaml file, an essential configuration file used by YOLOv8 to understand the dataset structure. This YAML file includes the paths to the training and validation image folders, the total number of classes, and the names of each vehicle category being detected. The accuracy and correctness of the data.yaml file are important because YOLOv8 relies on it to load the correct datasets and map predictions to labels.

Once this configuration is prepared, the process moves to the core phase: Train the YOLOv8 model. In this step, the YOLOv8 architecture is initialized using a pre-trained checkpoint (e.g., yolov8n.pt or yolov8s.pt), and the training process is executed over multiple epochs. The model learns to detect vehicles by adjusting its weights using backpropagation, minimizing loss functions, and improving detection accuracy with each iteration. During training, various performance metrics such as precision, recall, and mAP (mean Average Precision) are monitored to evaluate the learning progress.

After the model completes training, the next step is Evaluate the Model, where the trained YOLOv8 model is tested using the validation and test datasets. The model's performance is assessed based on its ability to correctly identify and localize vehicles in unseen images. This evaluation may involve generating confusion matrices, visualizing predicted bounding boxes, reviewing accuracy scores, and comparing performance across different classes. Any issues observed during this stage, such as overfitting, class imbalance, or low precision, may require fine-tuning or re-training the model. Once the evaluation confirms that the model meets the required accuracy standards, the process moves to the End stage, indicating successful completion of the model development phase.

Objective 3: Adaptive Signal Switching

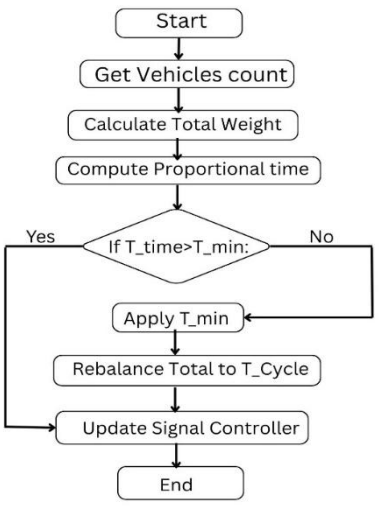


Fig. 4.5 Data Flow Diagram for Objective 3

The Signal Timing Algorithm Data Flow Diagram represents the internal logic used to compute adaptive green-light durations based on real-time vehicle density. The process begins at the Start node, initiating the dynamic traffic control cycle. The first major operation is Get Vehicles Count, where the system retrieves the number of vehicles detected in each lane using the trained YOLOv8 model. This input forms the foundation for the entire signal timing computation, as the accuracy of vehicle count directly influences the responsiveness of the traffic signal. Once vehicle counts are identified for all directions of the intersection, the workflow proceeds to the next step, Calculate Total Weight, where each lane's vehicle count is converted into a weighted value. These weights represent traffic pressure or demand on each lane, ensuring that lanes with heavier congestion receive higher priority.

Following the weight calculation, the algorithm enters the Compute Proportional Time stage. In this step, the total cycle time is mathematically divided among all lanes based on their proportional weights. This ensures that lanes with a higher number of vehicles receive a larger share of the available green time. For example, if one lane holds 60% of the total traffic load, it should ideally receive 60% of the available green duration. This step embodies the core principle of adaptive signalling smartly allocating time according to real-time traffic density rather than relying on fixed timers.

Once the proportional times are computed, the algorithm checks each allocated green duration and moves to Apply T-min (minimum green). In this critical step, the algorithm ensures that no lane receives a green phase shorter than a predefined minimum value, preventing situations where a lane receives too little green time for vehicles to start moving safely. This constraint guarantees usability and safety at the intersection, especially for low-traffic lanes or lanes with sudden drops in vehicle count. If the proportional allocation results in any lane falling below this threshold, the system automatically adjusts the duration upward to the T-min value.

However, increasing some green times can disrupt the total cycle duration, so the next step is to Rebalance Total to T-Cycle ensures that the cumulative green times, including adjustments, still fit within the defined total cycle length. During this rebalancing process, excess time may be deducted proportionally from other lanes, or the remaining available time may be redistributed across lanes with higher demand. This ensures the algorithm maintains the overall timing constraints while still being responsive to real-time traffic variations.

Objective 4: Traffic Simulation & System Evaluation

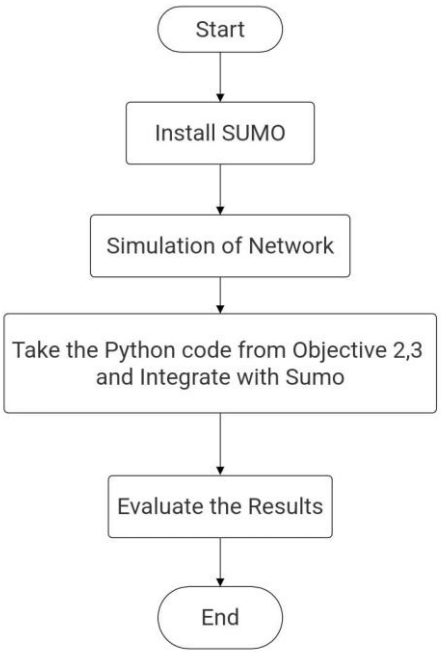


Fig. 4.6 Data Flow Diagram for Objective 4

The Data Flow Diagram illustrates the complete workflow involved in setting up, executing, and evaluating the SUMO-based simulation for the adaptive traffic signal control system. The process begins at the Start stage, where the simulation pipeline is initialized and all necessary configurations and dependencies are prepared. The first major step is Install SUMO, which involves downloading and setting up the SUMO (Simulation of Urban Mobility) framework on the local system. At this stage, all core SUMO components such as netedit, sumo-gui, netconvert, and duarouter are installed. These components are critical because they allow the user to design the road network, configure traffic routes, visualize traffic flow, and run the simulation in real-time. In addition to installing SUMO, environment variables must be configured properly so that Python scripts can interact with SUMO using the TraCI API. Ensuring a successful installation is crucial because SUMO acts as the simulation environment where the adaptive traffic control algorithm will be tested and validated.After installation, the workflow progresses to Simulation of Network, where the road intersection or multi-lane traffic network is created and executed. In this phase, the road topology is designed using SUMO's NetEdit tool, where lanes, intersections, traffic lights, entry points, exit points, and turning relationships are defined. Once the network design is completed, vehicle mobility patterns are generated using a route file (.rou.xml), which specifies how many vehicles enter each lane, at what intervals, and with what vehicle types.

The next stage in the workflow is the most critical: Take the Python code from Objective 2 & 3 and integrate with SUMO. Objective 2 typically represents the YOLOv8-based vehicle detection model, while Objective 3 corresponds to the adaptive signal timing algorithm. During integration, Python's TraCI library (Traffic Control Interface) is used to establish real-time communication between the traffic detection logic and the SUMO simulation engine. TraCI allows Python scripts to retrieve live vehicle counts from SUMO, modify traffic light states, update green times dynamically, and send the AI-generated signal timing decisions back into the simulation. This step effectively links the perception module (vehicle detection), the decision-making module (adaptive timing algorithm), and the environment module (SUMO). As SUMO simulates vehicles moving along the road network, the Python code continuously interacts with the simulation at each time step, applying new green times, reading lane density, and adjusting the signal cycle proportionally. This integration transforms SUMO from a static simulator into an intelligent, feedback-based traffic management system.

Once the Python code is fully integrated, the workflow moves to the Evaluate the Results stage. In this step, SUMO generates various performance metrics such as average waiting time, lane-wise queue length, number of stops, vehicle travel time, and intersection throughput. These metrics are recorded through TraCI, log files, or SUMO's built-in output gathering tools. The evaluation process compares the performance of the fixed-time traffic signal system with the AI-driven adaptive system to determine whether the adaptive approach successfully reduces congestion and improves traffic efficiency. Visualizations may be created using Matplotlib, Seaborn, or SUMO-GUI to show how traffic flow changes with adaptive timings. Observing improvements such as reduced waiting time, shorter queue lengths, and smoother flow provides quantitative evidence of the system's effectiveness. If necessary, multiple simulation runs are performed under low, medium, and peak-hour traffic conditions to verify robustness and reliability. Finally, the workflow reaches the End stage, marking the completion of the simulation and performance evaluation process. By this point, the entire system from vehicle detection to dynamic signal adjustment has been tested within a controlled, realistic simulation environment. The insights gathered from this module guide further fine-tuning of the algorithm or model retraining if improvements are needed. Overall, this Data Flow Diagram captures the essential phases of installing SUMO, simulating traffic, integrating adaptive control logic, and evaluating the effectiveness of the AI-driven system.

4.3 Requirement Analysis

Requirement analysis helps identify what the AI-based traffic signal system must achieve and what resources are needed to build it. The primary functional requirement is the ability to detect vehicles in real time using simulated camera inputs and accurately count them in each lane. The system must analyze traffic density, predict congestion, and dynamically adjust signal timings to improve traffic flow**.**

4.3.1 Software Requirements

Programming Languages and Frameworks:

- **Python:**
  Used for developing the core vehicle detection logic, traffic density estimation algorithms, adaptive signal control logic, and integration with the SUMO simulation environment. Python's wide range of libraries makes it ideal for computer vision and machine learning tasks.
- **OpenCV:**
  A powerful computer vision library essential for video frame processing, vehicle detection, object tracking, and pre-processing of traffic images. It helps extract real-time traffic density from live camera feeds or recorded videos.
- **TensorFlow/Keras or PyTorch:**
  High-level deep learning frameworks used to build, train, and evaluate Convolutional Neural Networks (CNNs) or object detection models (e.g., YOLO, SSD) for accurate vehicle recognition and classification. They automate feature extraction and enable real-time detection performance.
- **SUMO:**
  An open-source traffic simulation tool used to design road networks, simulate vehicle flow, and evaluate the adaptive traffic signal algorithm under realistic conditions. SUMO enables testing without real-world deployment.
- **Jupyter Notebook / VS Code / PyCharm:**
  Integrated development environments (IDEs) for writing, testing, and debugging Python scripts, deep learning models, and simulation integration code.

Simulation Software Requirements:

- **SUMO (Simulation of Urban Mobility):**
  Primary software for simulating road networks, traffic flow, signaling behavior, and vehicle movement.
- **NetEdit:**
  Required for building, editing, and configuring intersections, lanes, traffic lights, and road networks.
- **SUMO Tools (sumo-gui, netconvert, duarouter):**
  Used for route generation, network conversion, visualization, and controlling simulation parameters.
- **Python–SUMO Integration (TraCI):**
  Traffic Control Interface for real-time communication between Python scripts and SUMO simulation.

Annotation Tools

- **Label Studio:**
  A powerful open-source annotation tool used to label vehicles in images and videos. It supports bounding-box annotation, object classification, and integrates easily with machine learning pipelines.
- **Roboflow Annotate:**
  A web-based annotation and dataset management tool. Useful for:
  – Manually labeling vehicles
  – Data augmentation (cropping, flipping, brightness adjustment)
  – Splitting datasets into train/test/validation sets
  – Exporting annotations in YOLO, COCO, Pascal VOC formats.
- **LabelImg:**
  A lightweight, offline annotation tool widely used for creating bounding-box labels. It generates Pascal VOC (.xml) or YOLO (.txt) annotation files needed for model training.
- **CVAT:** CVAT (Computer Vision Annotation Tool) is an open-source, web-based annotation platform developed by Intel for creating high-quality datasets for computer vision applications.

4.3.2 Hardware Requirements

- **Processor:**

  Intel Core i5/i7 or AMD Ryzen 5/7 (4–8 cores)

  Provides faster processing for larger networks and real-time adaptive control.
- **RAM:**

  8 GB – 16 GB

  Ensures stable performance when running SUMO along with Python, OpenCV, and deep learning inference.
- **Storage:**

  10–20 GB free space

  For simulation files, video data, model weights, scripts, and logs.
- **Graphics:**

  NVIDIA GPU (Optional but Helpful)
  - GTX 1650 or higher
  - Used only if training or running deep learning models alongside SUMO. SUMO itself does not use GPU, but the AI model does.

Chapter 5

IMPLEMENTATION

The implementation of the AI-powered adaptive traffic signal system involves integrating computer vision, deep learning, and traffic simulation to create an intelligent, data-driven traffic management solution. The process begins with preparing and annotating a large vehicle dataset, followed by training a YOLOv8 model for accurate vehicle detection. The system then uses these detections to compute adaptive green-light timings through a proportional timing algorithm. SUMO simulation, connected via TraCI, enables real-time testing of the controller under realistic traffic conditions. Together, these components form a complete feedback-driven pipeline that dynamically adjusts signal timings based on live traffic density. This implementation ensures improved traffic flow, reduced congestion, and smarter signal control behavior.

Here is the step-by-step implementation guide for setup and a high-level breakdown of the code's logic.

5.1 Setup and Running Instructions

To run the Adaptive Traffic Signal System, you must first configure your Python environment, install the required libraries, and then execute the YOLO inference and SUMO simulation scripts.

Step 1: Install Dependencies

Open your terminal or command prompt and install the required Python libraries. The application uses a large number of packages for advanced processing.

```
# Create and activate a virtual environment
python -m venv venv
# Windows
venv\Scripts\activate
# Install required packages
pip install opencv-python numpy pandas matplotlib seaborn ultralytics
pip install torch torchvision --index-url https://download.pytorch.org/whl/cu118
pip install sumolib traci shapely albumentations
pip install roboflow label-studio-client
```

Step 2: Prepare the Dataset

- Download traffic datasets from Kaggle / Roboflow
- Annotate using CVAT / LabelImg / Roboflow Annotate
- Export in YOLO format
- Create the folder structure:

dataset/
images/train
images/val
labels/train
labels/val

- Create the data.yaml file containing class names and dataset paths.

Step 3: Training the YOLOv8 Model

To train the vehicle detection model used in the adaptive traffic signal system, the YOLOv8 (You Only Look Once) object detection framework was selected due to its high speed, accuracy, and suitability for real-time applications. The training process involves preparing the dataset, configuring the model, initiating the training procedure, and obtaining the final optimized model weights. The following steps outline the complete implementation.

Preparing the Training Environment

Before starting the training process, the Python environment must be properly configured with all required deep learning packages. The Ultralytics YOLO framework is installed along with PyTorch, which provides GPU acceleration for faster training. A separate virtual environment is recommended to keep dependencies organized.

pip install ultralytics

pip install torch torchvision

Once installed, YOLOv8 can be imported and used directly within Python scripts or executed using terminal commands.

Dataset Organization and Configuration

The vehicle images collected and annotated during the Data Preparation stage are arranged into a data.yaml configuration file is then created to define dataset paths, number of classes, and class names. An example format is shown below:

train: dataset/images/train

val: dataset/images/val

This file allows YOLOv8 to correctly load the dataset and map label IDs to their respective classes during training.

Starting the Training Process

Training can be initiated using a single Ultralytics command. The model is trained using the yolov8s.pt architecture, which offers a balance between accuracy and speed, making it suitable for real-time traffic applications.

yolo detect train data=data.yaml model=yolov8s.pt epochs=50 imgsz=640 batch=16

Explanation of parameters:

- data=data.yaml: Specifies dataset configuration.
- model=yolov8s.pt: Loads the YOLOv8 small model as a base.
- epochs=50: Number of training cycles.
- imgsz=640: Image resolution used during training.
- batch=16: Number of images processed at a time.

Throughout training, YOLOv8 automatically logs performance metrics such as precision, recall, and mean Average Precision (mAP). These metrics help evaluate how well the model is learning to detect vehicles.

Model Output and Results

After training completes, YOLOv8 generates the following output directory:

runs/train/<experiment_name>/

```
├── weights/
│     ├── best.pt
│     ├── last.pt
├── results.png
├── training_logs.csv
```

- best.pt is the most important file it contains the optimized model weights that achieved the highest validation accuracy.
- last.pt is the final model from the last epoch.
- results.png provides visual summaries of training progress, including loss curves and mAP improvements.

The best.pt file is used in the Signal Control module to perform real-time vehicle detection for adaptive traffic management.

Evaluation of the Trained Model

Once the model is trained, it is evaluated using the validation set. YOLOv8 provides built-in evaluation tools to measure its performance on unseen data:

yolo detect val model=best.pt data=data.yaml imgsz=640

This evaluation reports accuracy metrics and allows visualization of detection results, confirming whether the model is robust enough for integration into the live traffic control system.

Integration with the System

After validating the model performance, the best.pt file is integrated into the main system pipeline, where it is used to detect vehicles from either live camera feeds or SUMO-generated simulation frames. The detected vehicle counts are then passed to the timing algorithm to compute adaptive signal durations.

Step 4: Signal Switching Algorithm

This function calculates adaptive green-light durations based on the number of vehicles in each lane. First, it allocates time proportionally to each lane according to its traffic volume. Next, it ensures fairness by enforcing a minimum green time for all lanes. If the total exceeds the cycle time, durations are scaled down; if less, the remaining time is redistributed to busier lanes. The function then identifies the priority order by sorting lanes from highest to lowest vehicle count. Finally, it returns both the computed green times and the priority sequence for phase execution**.**

Code:

```
def allocate_and_prioritize(vehicle_counts, T_cycle=120, T_min=10):
    n = len(vehicle_counts)
    W_total = sum(vehicle_counts)
 # Step 1: Initial allocation
    green_times = [(v / W_total) * T_cycle if W_total > 0 else T_min for v in
      vehicle_counts]
 # Step 2: Apply minimum
    for i in range(n):
        if green_times[i] < T_min:
            green_times[i] = T_min
  # Step 3: Balance
    total_time = sum(green_times)
    if total_time > T_cycle:  # scale down
```

```
        scale = T_cycle / total_time
green_times = [max(T_min, g * scale) for g in green_times]
        elif total_time < T_cycle:  # redistribute
            remaining = T_cycle - total_time
            weights = [vehicle_counts[i] for i in range(n) if green_times[i] > T_min]

            W_extra = sum(weights)
                for i in range(n):
                    if green_times[i] > T_min:
                        green_times[i] += (vehicle_counts[i] / W_extra) * remaining

            # Step 4: Priority order (highest count first)
            priority_order = sorted(range(n), key=lambda i: vehicle_counts[i], reverse=True)

            return green_times, priority_order
    allocate_and_prioritize([30,40,20,12])
```

5.2 Simulation

The first step in the SUMO simulation process is building the road network, and this begins with defining all the road segments, known as *edges*. The file edges.xml contains the complete list of incoming and outgoing edges that form the structure of the intersection. Each edge represents a directional road segment connecting one node to another. For example, edges such as north_in, south_in, east_in, and west_in describe the roads entering the junction, while north_out, south_out, east_out, and west_out represent the outgoing roads.

Code:

```
<edges>
<!-- Incoming -->
<edge id="north_in" from="north" to="center" numLanes="2" speed="13"/>
<edge id="south_in" from="south" to="center" numLanes="2" speed="13"/>
<edge id="east_in" from="east" to="center" numLanes="2" speed="13"/>
<edge id="west_in" from="west" to="center" numLanes="2" speed="13"/>
```

```
<!-- Outgoing -->
<edge id="north_out" from="center" to="north" numLanes="2" speed="13"/>
<edge id="south_out" from="center" to="south" numLanes="2" speed="13"/>
<edge id="east_out" from="center" to="east" numLanes="2" speed="13"/>
<edge id="west_out" from="center" to="west" numLanes="2" speed="13"/>
</edges>
```

The second step in the SUMO simulation workflow is creating the connections file, which defines how vehicles move from one edge to another inside the intersection. While the edges file specifies the roads themselves, the connections file establishes all possible turning movements between incoming and outgoing roads.

In the file connections.xml, each <connection> element describes a legal maneuver for vehicles:

- Straight movement (dir="s")
- Left turn (dir="l")
- Right turn (dir="r")

Code:

```
<connections>
<!-- NORTH → -->
<connection from="north_in" to="south_out" priority="1" dir="s" state="M"/>
<connection from="north_in" to="east_out" priority="1" dir="l" state="M"/>
<connection from="north_in" to="west_out" priority="1" dir="r" state="M"/>
<!-- EAST → -->
<connection from="east_in" to="west_out" priority="1" dir="s" state="M"/>
<connection from="east_in" to="south_out" priority="1" dir="l" state="M"/>
<connection from="east_in" to="north_out" priority="1" dir="r" state="M"/>
<!-- SOUTH → -->
<connection from="south_in" to="north_out" priority="1" dir="s" state="M"/>
<connection from="south_in" to="west_out" priority="1" dir="l" state="M"/>
<connection from="south_in" to="east_out" priority="1" dir="r" state="M"/>
<!-- WEST → -->
<connection from="west_in" to="east_out" priority="1" dir="s" state="M"/>
<connection from="west_in" to="north_out" priority="1" dir="l" state="M"/>
<connection from="west_in" to="south_out" priority="1" dir="r" state="M"/>
</connections>
```

The third step in the SUMO simulation setup is creating the nodes file, which defines all the intersection points (junctions) in the road network. While edges describe the roads and connections describe possible vehicle movements, the nodes file establishes the exact geographical coordinates and behaviour of each junction.

In the file nodes.xml, each <node> represents a key point in the intersection layout:

- The central junction is defined as

  node id="center" type="traffic_light"

  This indicates that the intersection is signal-controlled and will later be linked to a traffic light logic file.
- Four additional nodes represent the entry and exit points of the intersection:
  - north at coordinates (0, 50)
  - south at (0, -50)
  - east at (50, 0)
  - west at (-50, 0)

These coordinates determine the physical structure of the map and define where incoming and outgoing edges originate. SUMO uses these node positions to compute the geometric shape of each road and to determine how vehicles navigate the junction.

Code:

```
<nodes>
<node id="center" x="0" y="0" type="traffic_light"/>
<node id="north" x="0" y="50"/>
<node id="south" x="0" y="-50"/>
<node id="east" x="50" y="0"/>
<node id="west" x="-50" y="0"/>
</nodes>
```

The fourth step of the simulation involves integrating a Python-based controller that runs the traffic logic, communicates with SUMO using TraCI, and compares the performance of two systems: a fixed-time baseline signal plan and the adaptive AI-driven signal plan. The file adaptive_compare.py serves as the core execution engine for this step. It controls the traffic lights, collects queue statistics, runs YOLO-based vehicle detection (optional), logs data, and generates a performance comparison dashboard.

In this script, SUMO is launched through TraCI, and the traffic light node (center) is assigned an 8-phase program to support independent movements for all four directions north, east, south, and west. The baseline system uses a predefined timing pattern in which each phase receives a fixed green duration of 15 seconds followed by a 3-second yellow phase. This allows SUMO to simulate normal, non-adaptive traffic behaviour.

The adaptive mode dynamically selects which direction receives green time based on real-time traffic density from SUMO's built-in vehicle counters. Optionally, YOLOv8 detection is fused with SUMO counts to improve accuracy. The algorithm identifies active approaches, builds a fairness-based cycle order, selects the lane group with the highest demand, and assigns a calculated green time with minimum and maximum limits to avoid starvation or excessive allocation.

During the simulation, the script writes performance statistics into CSV files performance_baseline.csv and performance_ai.csv logging queue lengths, selected phases, green times, and YOLO counts. The controller also saves a deeper debug log for analysis and generates a comparison dashboard with bar charts to visualize improvements in average queue lengths. After completing both modes, the script outputs a detailed console summary highlighting total vehicles, average queues, green time distribution, and the percentage improvement achieved by the AI-based system.

This step is essential because it brings together all previous components the network structure, routing, and traffic lights and adds the intelligence layer required to evaluate how adaptive control improves traffic conditions over fixed-time systems. It forms the computational backbone of the simulation and provides measurable results for performance comparison.

Code:

```
import os          # filesystem
import time        # sleep/delay
import csv         # csv writing
import random      # random operations
import traceback   # debug trace
try:
    import traci                    # SUMO TraCI API
    from traci import trafficlight        # traffic light interface
except:
    raise RuntimeError("Install SUMO & set SUMO_HOME")    # SUMO required
import pandas as pd                    # data analysis
```

```
import matplotlib.pyplot as plt          # plotting
SUMO_BINARY = "sumo-gui"                 # SUMO GUI
SUMO_CONFIG = "simulation.sumocfg"       # SUMO config
NET_FILE = "network.net.xml"             # network file
ROUTE_FILE = "routes.rou.xml"            # route file

TLS_ID = "center"                        # traffic light ID
SLEEP_BETWEEN_STEPS = 0.001              # sleep time
YOLO_EVERY_N_STEPS = 10                  # YOLO frequency
MAX_STEPS = 1200                         # max sim steps
BASE_CSV = "performance_baseline.csv"    # baseline CSV
AI_CSV = "performance_ai.csv"            # AI CSV
DEBUG_CSV = "ai_cycle_debug.csv"         # debug CSV
DASH_PNG = "dashboard.png"               # dashboard image
APPROACH_EDGES = {                       # lane --> approach mapping
    "north": "north_in",
    "east": "east_in",
    "south": "south_in",
    "west": "west_in",
}
GUI_FAST_FILE = "gui_fast.xml"           # fast GUI settings
def ensure_gui_fast_file():              # ensure fast GUI XML exists
    if os.path.exists(GUI_FAST_FILE):
        return
    with open(GUI_FAST_FILE, "w") as f:
        f.write("""
<viewsettings>
    <delay value="0"/>
    <antialiasing value="false"/>
    <showLaneBorders value="false"/>
    <showLinkDecals value="false"/>
    <background value="0,0,0"/>
</viewsettings>
""")
```

```
def safe_traci_start():                # start SUMO with fast gui
ensure_gui_fast_file()
    traci.start([
        SUMO_BINARY, "-c", SUMO_CONFIG,
        "--start",
        "--quit-on-end",
        f"--gui-settings-file={GUI_FAST_FILE}"])

time.sleep(0.05)                # warm-up delay
USE_YOLO = True                        # yolo flag
YOLO_CLASSES_VEH = {2, 3, 5, 7}            # vehicle classes
_yolo = None
try:
    from ultralytics import YOLO           # YOLO import
    _yolo = YOLO("yolov8n.pt")             # load model
    print("YOLO Loaded")
except:
    USE_YOLO = False                   # disable if missing
    print("YOLO disabled")
def yolo_total_from_gui(view_id, snap="frame.png"):     # capture → detect
    if not USE_YOLO:
        return 0
    try:
        traci.gui.screenshot(view_id, snap)            # screenshot
        res = _yolo(snap, verbose=False)               # inference
        return sum(1 for box in res[0].boxes if int(box.cls[0]) in YOLO_CLASSES_VEH)
    except:
        return 0
def get_counts():                      # vehicle count on each approach
    q = {}
    for d, e in APPROACH_EDGES.items():
        try:
            q[d] = traci.edge.getLastStepVehicleNumber(e)
        except:
```

```
    q[d] = 0
    return q
def install_8_phase_tls(tls):              # install 8-phase logic
    try:
        lanes = traci.trafficlight.getControlledLanes(tls)
        if not lanes:
            return
        def approach(l): return l.split("_")[0]        # lane → approach
        phases = []
        dirs = ["north","east","south","west"]
        def make_state(g, col):                        # green/yellow states
            s = ""
            for ln in lanes:
                s += col if approach(ln) == g else "r"
            return s
        for g in dirs:
            phases.append(trafficlight.Phase(15, make_state(g, "G")))   # green
            phases.append(trafficlight.Phase(3, make_state(g, "y")))    # yellow
        logic = trafficlight.Logic("8phase", 0, 0, phases)             # install program
        traci.trafficlight.setProgramLogic(tls, logic)
    except:
        traceback.print_exc()
def baseline_run(max_steps=MAX_STEPS):       # baseline fixed-cycle
    print("\nRunning Baseline...")
    with open(BASE_CSV, "w", newline="") as f:
        csv.writer(f).writerow(["Step","North","East","South","West","GreenTime"])
    safe_traci_start()
    install_8_phase_tls(TLS_ID)
    step = 0
    phase = 0
    next_change = 0
    while step < max_steps and traci.simulation.getMinExpectedNumber() > 0:
        traci.simulationStep()              # run SUMO
        if step >= next_change:
```

```
traci.trafficlight.setPhase(TLS_ID, phase)
            dur = 15 if phase % 2 == 0 else 3
            traci.trafficlight.setPhaseDuration(TLS_ID, dur)
            next_change = step + dur
            phase = (phase + 1) % 8
        q = get_counts()
        with open(BASE_CSV, "a", newline="") as f:
            csv.writer(f).writerow([step,q["north"],q["east"],q["south"],q["west"],15])
        step += 1
    traci.close()
def adaptive_run(max_steps=MAX_STEPS):       # adaptive fairness + density
    print("\nRunning ADAPTIVE AI...")
    with open(AI_CSV, "w", newline="") as f:
        csv.writer(f).writerow([
            "Step","North","East","South","West",
            "SelectedGroup","YOLO_Total","Fused_Total","GreenTime"
        ])
    with open(DEBUG_CSV, "w", newline="") as f:
        csv.writer(f).writerow([
            "Step","ActiveGroups","CycleOrder","Selected","Vehicles","GreenTime"
        ])
    safe_traci_start()
    install_8_phase_tls(TLS_ID)
    step = 0
    view = traci.gui.getIDList()[0]          # gui id
    groups = ["north","east","south","west"]  # 4-approach
    cycle_list = []                          # fairness cycle
    cycle_index = 0
    rotation_count = 0
    group_to_phase = {"north":0,"east":2,"south":4,"west":6}     # green phases
    cur_group = "north"
    next_switch = 0
    MIN_G, MAX_G = 12, 45                                # green cap
    YEL = 3                                          # yellow time
```

```
while step < max_steps and traci.simulation.getMinExpectedNumber() > 0:
    traci.simulationStep()                    # SUMO step
    q = get_counts()
    active = [g for g in groups if q[g] > 0]          # non-empty approaches
    if len(active) == 0:
        step += 1
        continue
    if rotation_count < len(active):                  # fairness pass
        if not cycle_list:
cycle_list = sorted(active, key=lambda g: q[g], reverse=True)
            cycle_index = 0
        selected = cycle_list[cycle_index % len(cycle_list)]
        cycle_index += 1
        rotation_count += 1
    else:
        selected = max(active, key=lambda g: q[g])      # max-queue
        rotation_count = 0
        cycle_index = 0
        cycle_list = []
    base_time = q[selected] * 5.0                     # green scaling
    fairness = min(q.values()) * 2
    green = int(max(MIN_G, min(MAX_G, base_time + fairness)))
    yolo = yolo_total_from_gui(view) if USE_YOLO and step %
YOLO_EVERY_N_STEPS == 0 else 0
    fused = max(sum(q.values()), yolo)
    if step >= next_switch:                           # switching
        if step > 0:
            y_phase = group_to_phase[cur_group] + 1
            traci.trafficlight.setPhase(TLS_ID, y_phase)
            traci.trafficlight.setPhaseDuration(TLS_ID, YEL)
            traci.simulationStep()
        cur_group = selected
        traci.trafficlight.setPhase(TLS_ID, group_to_phase[cur_group])
        traci.trafficlight.setPhaseDuration(TLS_ID, green)
```

```
next_switch = step + green
        with open(AI_CSV, "a", newline="") as f:
            csv.writer(f).writerow([
                step,q["north"],q["east"],q["south"],q["west"],
                selected,yolo,fused,green])
        with open(DEBUG_CSV, "a", newline="") as f:
            csv.writer(f).writerow([
                step,active,cycle_list if cycle_list else "[]",
                selected,q[selected],green
            ])
        step += 1
    traci.close()
    print("AI Complete")
  def summarize(csv_file, label):              # simple stats
    df = pd.read_csv(csv_file)
    lanes = ["North","East","South","West"]
    df["Total"] = df[lanes].sum(axis=1)
    print(f"\nSummary {label}:", df["Total"].sum())
    return df["Total"].sum(), df[lanes].mean().to_dict()
def make_and_save_dashboard():               # bar chart
    df_b = pd.read_csv(BASE_CSV)
    df_a = pd.read_csv(AI_CSV)
    avg_b = df_b[["North","East","South","West"]].mean()
    avg_a = df_a[["North","East","South","West"]].mean()
    fig, ax = plt.subplots(figsize=(8,5))
    idx = range(4)
w = 0.35
    ax.bar([i-w/2 for i in idx], avg_b, width=w, label="Baseline")
    ax.bar([i+w/2 for i in idx], avg_a, width=w, label="AI")
    ax.set_xticks(idx)
    ax.set_xticklabels(["North","East","South","West"])
    ax.set_ylabel("Avg Queue")
    ax.set_title("Baseline vs AI")
    ax.legend()
```

```
plt.tight_layout()
    plt.savefig(DASH_PNG)
    plt.close()
def print_cmd_dashboard():                # text-based summary
    df_b = pd.read_csv(BASE_CSV)
    df_a = pd.read_csv(AI_CSV)
    avg_b = df_b[["North","East","South","West"]].mean().round(2)
    avg_a = df_a[["North","East","South","West"]].mean().round(2)
    total_b = int(df_b[["North","East","South","West"]].sum().sum())
    total_a = int(df_a[["North","East","South","West"]].sum().sum())
    green_b = round(df_b["GreenTime"].mean(), 2)
    green_a = round(df_a["GreenTime"].mean(), 2)
    improvement = 0
    if avg_b.sum() > 0:
        improvement = round((avg_b.sum() - avg_a.sum()) / avg_b.sum() * 100, 2)
    print("\n-------------------------------------------")
    print("         TRAFFIC SIGNAL PERFORMANCE       ")
    print("-------------------------------------------\n")
    print("BASELINE PERFORMANCE")
    print(f"• Total Vehicles: {total_b}")
    print(f"• Avg Queue:")
    print(f"    North: {avg_b['North']}")
    print(f"    East : {avg_b['East']}")
    print(f"    South: {avg_b['South']}")
    print(f"    West : {avg_b['West']}")
    print(f"• Avg Green Time: {green_b} sec\n")
    print("ADAPTIVE AI PERFORMANCE")
    print(f"• Total Vehicles: {total_a}")
    print("• Avg Queue:")
    print(f"    North: {avg_a['North']}")
    print(f"    East : {avg_a['East']}")
    print(f"    South: {avg_a['South']}")
    print(f"    West : {avg_a['West']}")
    print(f"• Avg Green Time: {green_a} sec\n")
```

```
    print("-----------------------------------------")
    print(f"      AI QUEUE REDUCTION: {improvement} %")
    print("-----------------------------------------\n")
def main():                          # main entry
    baseline_run()
    adaptive_run()
    print_cmd_dashboard()
    make_and_save_dashboard()
if __name__ == "__main__":
    main()
```

The fifth step in the simulation process introduces the file adaptive_main.py, which implements a real-time adaptive lane-by-lane traffic signal controller. This script interacts directly with the SUMO simulator using TraCI and continuously adjusts the signal phases based on the current lane congestion. Unlike a predefined cycle or a multi-phase signal plan, this controller dynamically selects the lane with the highest vehicle count and allocates green time proportional to its demand. This makes it one of the key components for demonstrating adaptive, AI-style traffic control.The script begins by launching SUMO with the specified network configuration (simulation.sumocfg) and identifying the traffic light to be controlled. SUMO provides a list of lanes supervised by the signal, and the controller extracts the associated lane order and green-phase indices. These indices are important because they map each lane to its corresponding green-phase in the traffic light logic defined in the network files.During every simulation run, the controller retrieves the current number of waiting vehicles in each lane using traci.lane.getLastStepVehicleNumber(). It then identifies the most congested lane and computes an appropriate green duration based on a simple adaptive formula:

$$\text{green\_time} = \text{MIN\_GREEN} + (\text{BASE\_PER\_VEHICLE} \times \text{vehicle\_count})$$

This duration is clamped between preset lower and upper bounds to maintain safety (e.g., a minimum of 8 seconds and a maximum of 45 seconds). Once calculated, the controller updates the traffic light phase using TraCI commands, holds that green phase for the computed duration, and then allows the simulation to progress naturally to the yellow phase.This script effectively converts traffic density into real-time control actions, demonstrating adaptive lane prioritization. It also includes several safety and robustness features such as fallback mappings, green-phase detection, simulation-step synchronization, and exception handling for

phase switching. Compared to the baseline fixed-time model, this approach greatly improves responsiveness during uneven traffic conditions by always prioritizing the lane with the highest demand.With this step completed, the simulation now fully supports live adaptive traffic signal behavior, making it possible to compare fixed-time and adaptive control performance using realistic SUMO traffic flows.

Code:

```
import time  # timing
import traci  # TraCI SUMO API
import os  # filesystem
SUMO_BINARY = "sumo-gui"  # GUI mode
SUMO_CONFIG = "simulation.sumocfg"  # config file
TLS_ID = "C"  # traffic light id
MIN_GREEN = 8  # minimum green time
MAX_GREEN = 45  # maximum green time
BASE_PER_VEH = 2  # seconds added per vehicle
def clamp(x, lo, hi): return max(lo, min(hi, x))  # limit x within range
def get_controlled_lanes(tls_id):  # fetch lanes controlled by TLS
    lanes = traci.trafficlight.getControlledLanes(tls_id)
    seen, out = set(), []
    for ln in lanes:
        if ln not in seen:
seen.add(ln)
            out.append(ln)
    return out
def count_vehicles_per_lane(lanes):  # count vehicles on each lane
    return {ln: traci.lane.getLastStepVehicleNumber(ln) for ln in lanes}
def compute_green_from_count(count):  # green = min + k*count
    return int(clamp(MIN_GREEN + BASE_PER_VEH * count, MIN_GREEN,
MAX_GREEN))
def find_green_phase_indices(tls_id):  # detect phases containing G
    tl = traci.trafficlight.getCompleteRedYellowGreenDefinition(tls_id)
    if not tl: return []
    phases = tl[0]["phases"]
    out = []
```

```
for i, p in enumerate(phases):
        if "G" in p.get("state", ""): out.append(i)
    return out
def run():  # main loop
    traci.start([SUMO_BINARY, "-c", SUMO_CONFIG, "--start"])
    print("SUMO+TraCI started")
    lanes = get_controlled_lanes(TLS_ID)
    if not lanes:
        print("No controlled lanes for TLS", TLS_ID)
 traci.close(); return
    green_idxs = find_green_phase_indices(TLS_ID)
    if not green_idxs:
        print("No green phases detected")
        traci.close(); return
    print("Controlled lanes:", lanes)
    print("Green phase indices:", green_idxs)
    sim_step = 0
    try:
        while traci.simulation.getMinExpectedNumber() > 0:
            traci.simulationStep()
            sim_step += 1
if sim_step % 5 == 0:  # reevaluate every 5 steps
                counts = count_vehicles_per_lane(lanes)
                best_lane = max(lanes, key=lambda ln: (counts[ln], -lanes.index(ln)))
                best_count = counts[best_lane]
                green_time = compute_green_from_count(best_count)
                idx = lanes.index(best_lane)
                phase = green_idxs[idx] if idx < len(green_idxs) else green_idxs[idx %
len(green_idxs)]
                print("\n--- STEP", sim_step, "---")
                print("Counts:", counts)
                print(f"Selected lane: {best_lane} -> phase {phase} -> {green_time}s")
                try:
                    traci.trafficlight.setPhase(TLS_ID, int(phase))  # set green
```

```
except Exception as ex:
                print("Phase set failed:", ex)
            for _ in range(green_time): traci.simulationStep()  # hold green
traci.simulationStep()  # allow yellow to apply
   finally:
      traci.close()
      print("Simulation finished")
if __name__ == "__main__":
   run()
```

The next step in the SUMO simulation workflow is to define the movement patterns of vehicles using a *routes file*. The file routes.rou.alt.xml contains all the vehicles that will appear in the simulation, their departure times, and the paths they will take across the network. This file is essential because SUMO requires predefined routes for every vehicle entering the traffic system. Without route definitions, the simulation cannot generate or move vehicles on the road network.In this step, the duarouter tool automatically generates realistic vehicle routes based on the defined network structure. Each <vehicle> entry includes a unique ID, a departure time in seconds, and a <routeDistribution> element that assigns the path for that vehicle. The routes are represented as a sequence of edges such as north_in → east_out, west_in → south_out, etc. These edges correspond to the incoming and outgoing lanes at each approach of the intersection. The costs and probabilities attached to each route are computed to ensure vehicles follow practical and shortest paths.

This file defines more than 250 vehicle entries, representing dynamic and continuous traffic arriving from all four directions. Each second, a new vehicle enters the system, creating a realistic level of congestion for evaluating both fixed-time signals and adaptive controllers. The presence of mixed turning movements straight, left, and right ensures that the traffic demand is diverse and mirrors real-world intersection behavior.

By creating this detailed routes file, SUMO can simulate how traffic flows through the junction under varying demand levels. The adaptive controller designed using YOLO vehicle counts will later use these routes to test and compare performance against fixed-time control. Thus, this step completes the preparation of the simulated traffic demand and forms the basis for running the full-scale intersection simulation.

The sixth step in the simulation process uses the helper script probe_ids.py to verify and inspect the elements of the SUMO network before running the full adaptive or baseline traffic controller. This probing step is essential to ensure that all node IDs, lane IDs, edge IDs, and traffic light IDs are correctly generated and recognized by SUMO.

If any name mismatches or missing IDs exist, the adaptive controller would fail later when attempting to access or modify the traffic signals.

In this script, SUMO is launched in GUI mode using the specified configuration file (simulation.sumocfg). Once SUMO is running, the script establishes a TraCI (Traffic Control Interface) connection. Through TraCI, the script queries and prints the list of all active network components, including:

- Traffic Light IDs
- Edge IDs (first 10 shown for verification)
- Lane IDs (first 12 shown for verification)

These printed outputs allow the user to confirm the exact names and ordering of lanes that SUMO uses internally. This is particularly important because the adaptive controller relies on the ability to map each lane to a specific signal phase. If IDs are incorrect, inconsistent, or different from the expected naming pattern, the controller cannot properly assign green times or retrieve lane vehicle counts.

After completing the inspection, the script closes the TraCI connection, confirming that SUMO is responding correctly and that all network elements have loaded without errors. This probing step ensures that the network is correctly assembled, free from structural issues, and ready for integration with the real-time adaptive control system.

Code:

```
import traci
SUMO_BINARY = "sumo-gui"
SUMO_CONFIG = "simulation.sumocfg"
# start SUMO (GUI) and connect
traci.start([SUMO_BINARY, "-c", SUMO_CONFIG, "--start"])
print("SUMO started")
print("Traffic light IDs:", traci.trafficlight.getIDList())
print("Edge IDs (first 10):", traci.edge.getIDList()[:10])
print("Lane IDs (first 12):", traci.lane.getIDList()[:12])
traci.close() ,print("Done.")
```

The seventh step in the simulation process involves reviewing the auto-generated connection file, named prefix=plain.con.xml. This file is created by SUMO's netconvert tool when the network is built from the user-defined nodes, edges, and manual connections. While the earlier connections.xml file describes the basic turning movements at the intersection, this plain connection file contains the fully resolved, lane-level connections that SUMO produces after processing the entire network. This makes it one of the most important internal files for ensuring that vehicles can move smoothly and safely through the junction.

Code:

```
<connections xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" version="1.20" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/
connections_file.xsd">
<connection from="east_in" to="south_out" fromLane="0" toLane="0"/>
<connection from="east_in" to="west_out" fromLane="0" toLane="0"/>
<connection from="east_in" to="west_out" fromLane="1" toLane="1"/>
<connection from="east_in" to="north_out" fromLane="1" toLane="1"/>
<connection from="east_out" to="east_in" fromLane="1" toLane="1"/>
<connection from="north_in" to="east_out" fromLane="0" toLane="0"/>
<connection from="north_in" to="south_out" fromLane="0" toLane="0"/>
<connection from="north_in" to="south_out" fromLane="1" toLane="1"/>
<connection from="north_in" to="west_out" fromLane="1" toLane="1"/>
<connection from="north_out" to="north_in" fromLane="1" toLane="1"/>
<connection from="south_in" to="west_out" fromLane="0" toLane="0"/>
<connection from="south_in" to="north_out" fromLane="0" toLane="0"/>
<connection from="south_in" to="north_out" fromLane="1" toLane="1"/>
<connection from="south_in" to="east_out" fromLane="1" toLane="1"/>
<connection from="south_out" to="south_in" fromLane="1" toLane="1"/>
<connection from="west_in" to="north_out" fromLane="0" toLane="0"/>
<connection from="west_in" to="east_out" fromLane="0" toLane="0"/>
<connection from="west_in" to="east_out" fromLane="1" toLane="1"/>
<connection from="west_in" to="south_out" fromLane="1" toLane="1"/>
<connection from="west_out" to="west_in" fromLane="1" toLane="1"/>
</connections>
```

The eighth step in the SUMO simulation pipeline involves the network conversion configuration file, named prefix=plain.netccfg. This file is automatically generated by netconvert, SUMO's network-building tool, and it plays a central role in defining how the raw input files (nodes,

edges, and connections) are processed to create the final, executable SUMO network.

Code:

```
<netconvertConfiguratioxmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/netconvertConfiguration.xsd">
<input>
      <node-files value="prefix=plain.nod.xml"/>
      <edge-files value="prefix=plain.edg.xml"/>
      <connection-files value="prefix=plain.con.xml"/>
      <tllogic-files value="prefix=plain.tll.xml"/>
   </input>
   <output>
      <output-file value="network.net.xml"/>
   </output>
  <processing>
      <lefthand value="true"/>
   </processing>
</netconvertConfiguration>
```

The ninth step in the SUMO simulation pipeline involves the automatically generated traffic light logic file, named prefix=plain.tll.xml. This file is produced by netconvert when the network is built, and it defines the default, static traffic signal plan for the intersection before any adaptive or Python-controlled logic is applied. SUMO creates this file by analyzing the network geometry, the number of incoming lanes, and the legal turning movements described in the connections file.

Inside the file, a single <tlLogic> block is assigned to the junction with ID "center", which corresponds to your main four-arm intersection. The traffic light is configured in static mode, meaning phases operate on a fixed schedule unless overridden by an external controller (like your adaptive Python script). The program consists of four phases: two full green phases and two yellow (transition) phases. The first green phase (duration="42") gives right-of-way to the east–west direction, while the second (duration="42") gives right-of-way to the north–south direction. Each green phase is followed by a 3-second yellow phase to allow vehicles to

clear the intersection safely.The "state" strings such as GGGgrrrrGGGgrrrr represent the signal condition of every lane-to-lane connection, with characters indicating green (G), yellow (y), or red (r). These sequences correspond to the 16 connections listed below the phase definitions, where each connection includes its linkIndex that maps directly to a position in the state string. SUMO uses this mapping to determine exactly which turning movements receive green, yellow, or red in each phase.

Code:

```
<tlLogics xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance" version="1.20" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/
tllogic_file.xsd">
<tlLogic id="center" type="static" programID="0" offset="0">
<phase duration="42" state="GGGgrrrrGGGgrrrr"/>
<phase duration="3" state="yyyyrrrryyyyrrrr"/>
<phase duration="42" state="rrrrGGGgrrrrGGGg"/>
<phase duration="3" state="rrrryyyyrrrryyyy"/>
</tlLogic>
<connection from="east_in" to="south_out" fromLane="0" toLane="0" tl="center" li
nkIndex="4"/>
<connection from="east_in" to="west_out" fromLane="0" toLane="0" tl="center" li
nkIndex="5"/>
<connection from="east_in" to="west_out" fromLane="1" toLane="1" tl="center" li
nkIndex="6"/>
<connection from="east_in" to="north_out" fromLane="1" toLane="1" tl="center" li
nkIndex="7"/>
<connection from="north_in" to="east_out" fromLane="0" toLane="0" tl="center" li
nkIndex="8"/>
<connection from="north_in" to="south_out" fromLane="0" toLane="0" tl="center"
linkIndex="9"/>
<connection from="north_in" to="south_out" fromLane="1" toLane="1" tl="center"
linkIndex="10"/>
<connection from="north_in" to="west_out" fromLane="1" toLane="1" tl="center" l
inkIndex="11"/>
<connection from="south_in" to="west_out" fromLane="0" toLane="0" tl="center" l
inkIndex="0"/>
```

```
<connection from="south_in" to="north_out" fromLane="0" toLane="0" tl="center"
linkIndex="1"/>
<connection from="south_in" to="north_out" fromLane="1" toLane="1" tl="center"
linkIndex="2"/>
<connection from="south_in" to="east_out" fromLane="1" toLane="1" tl="center" li
nkIndex="3"/>
<connection from="west_in" to="north_out" fromLane="0" toLane="0" tl="center" l
inkIndex="12"/>
<connection from="west_in" to="east_out" fromLane="0" toLane="0" tl="center" li
nkIndex="13"/>
<connection from="west_in" to="east_out" fromLane="1" toLane="1" tl="center" li
nkIndex="14"/>
<connection from="west_in" to="south_out" fromLane="1" toLane="1" tl="center" l
inkIndex="15"/>
</tlLogics>
```

The tenth step in the SUMO simulation workflow involves customizing the appearance and performance of the SUMO graphical user interface using the file gui_fast.xml. Although not required for the core logic of the simulation, this file ensures that SUMO runs smoothly, renders quickly, and eliminates unnecessary visual elements. This is essential when performing long-duration simulations, adaptive control experiments, or exporting demonstration videos.

The file consists of a <viewsettings> block that adjusts various GUI parameters. It sets the simulation delay to **0**, allowing SUMO to run in real time or as fast as possible, depending on system performance. Antialiasing is disabled to reduce graphical overhead, providing faster rendering during experiments. Visual details such as lane borders and link decals are also turned off, making the scene cleaner and improving display performance. Additionally, the background color is set to black (0,0,0), which helps highlight vehicles and signal states more clearly.These simple but important adjustments ensure that the GUI remains efficient even when running heavy experiments, especially when integrating external Python scripts for adaptive traffic control. While the SUMO network and controllers operate independently of this file, the visual settings make simulation observation smoother, faster, and less resource-intensive.

In summary, gui_fast.xml optimizes the SUMO display environment for rapid simulation execution, reducing graphical load and improving visual clarity during testing of both fixed.

Code:

```
<viewsettings>
<delay value="0"/>
<antialiasing value="false"/>
<showLaneBorders value="false"/>
<showLinkDecals value="false"/>

<background value="0,0,0"/>
</viewsettings>
```

The eleventh step in the SUMO simulation process is the creation of the main simulation configuration file, named simulation.sumocfg. This is one of the most important components of the entire simulation because it acts as the central controller that ties together all other files network, routes, traffic signals, GUI settings, and external controller scripts into a single executable simulation.

The configuration file consists of several structured blocks that define how SUMO should run the simulation. The <input> section references the core network file (network.net.xml) and the routes file (routes.rou.alt.xml), ensuring that SUMO loads the complete road layout and all vehicle paths. Optional input files like the GUI configuration (gui_fast.xml) can also be included to improve display performance.The <time> section specifies the simulation time range, including the start and end times, which determine how long the simulation will run. The <processing> section controls performance-related settings such as step-length (simulation tick duration), lane-changing models, or collision checks. Meanwhile, the <report> section manages output logs, error messages, and summary statistics, allowing you to collect performance data for analysis.

This file also enables interaction with external controllers through TraCI. When SUMO is launched with the --remote-port argument, simulation.sumocfg ensures that the network is loaded and ready for the Python adaptive controller (adaptive_main.py or adaptive_compare.py) to connect and modify traffic signal states in real time.In summary, simulation.sumocfg serves as the master configuration that orchestrates the entire SUMO environment. It seamlessly integrates all the elements created in earlier steps nodes, edges, connections, routes, detectors, traffic light logic, and GUI settings allowing the adaptive AI-driven traffic controller to operate on a complete and well-structured simulation model.

Code:

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <input>

<net-file value="network.net.xml"/>
<route-files value="routes.rou.xml"/>
   </input>
   <time>
      <begin value="0"/>
      <end value="500"/>
   </time>
</configuration>
```

Frontend Development:

```
import streamlit as st  # Streamlit UI
import subprocess  # run simulation scripts
import pandas as pd  # dataframes
import os  # filesystem
import time  # timing
import plotly.graph_objs as go  # plotting
import plotly.express as px  # plotting shortcuts
# ---------------- Config ----------------
BASE_DIR = r"C:\Users\Mokshitha Thota\Documents\projects\AI POWERED TSP\SendAnywhere_746655\AdaptiveTrafficNEW"  # project folder
UPLOADED_SCRIPT_PATH = os.path.join(BASE_DIR, "adaptive_main.py")  # adaptive script path
PERF_AI = os.path.join(BASE_DIR, "performance_ai.csv")  # AI output
PERF_BASE = os.path.join(BASE_DIR, "performance_baseline.csv")  # baseline output
AI_CYCLE = os.path.join(BASE_DIR, "ai_cycle_debug.csv")  # debug CSV
SIM_SCRIPT_MAIN = os.path.join(BASE_DIR, "adaptive_main.py")  # main sim script
SIM_SCRIPT_COMPARE = os.path.join(BASE_DIR, "adaptive_compare.py")  # optional compare script
FRAME_PNG = os.path.join(BASE_DIR, "frame.png")  # snapshot path
FRAME_PNG_MNT = "/mnt/data/frame.png"  # container fallback (not used now)
st.set_page_config(page_title="AI Traffic Signal   Analytics", layout="wide",
```

```
initial_sidebar_state="collapsed")  # page config
# ---------- Small CSS ----------
st.markdown("""
   <style>
   .controls-row { display:flex; gap:14px; align-items:center; justify-content:flex-start; margin-bottom:12px; }
  .control-btn { border-radius:12px; padding:10px 16px; font-weight:700; }
   .kpi-grid { display:grid; grid-template-columns: repeat(3, 1fr); gap: 12px; margin-bottom:12px; }
   .kpi-card { background: linear-gradient(90deg,#ffecd2,#fcb69f); padding:12px; border-radius:10px; box-shadow:0 8px 20px rgba(15,25,40,0.06); }
   </style>
   """, unsafe_allow_html=True)  # CSS for cards
# ---------- Helpers ----------
def read_csv_safe(path):  # read csv with fallbacks
   if not os.path.exists(path):
      return None
   try:
      return pd.read_csv(path, encoding="utf-8", engine="python")
   except Exception:
      try:
         return pd.read_csv(path, encoding="latin1", engine="python")
      except Exception:
         return None
def fmt(x):  # format numeric to 2 decimals
   try:
      return f"{float(x):.2f}"
   except Exception:
      return "-"
def compute_total_vehicles_from_df(df):  # compute total throughput from df
if df is None:
      return '-'
   cols = {c.lower(): c for c in df.columns}
   # throughput column present
   if 'throughput' in cols:
      try:
return int(df[cols['throughput']].sum())
```

```
except Exception:
            pass
    # wide format (north/east/south/west)
if all(k in cols for k in ['north', 'east', 'south', 'west']):
        try:
total = int(df[[cols['north'], cols['east'], cols['south'], cols['west']]].sum(axis=1).sum())
return total
        except Exception:
            pass
    return '-'
# --- Safe rerun wrapper (covers multiple Streamlit versions) ---
def safe_rerun():  # trigger app rerun across Streamlit versions
    """Try to trigger a rerun across different Streamlit versions.
    Order of attempts:
      1. st.experimental_rerun() if exposed
      2. raise internal RerunException() if available
      3. fallback: set a session flag and call st.stop()
    """
    try:
        st.experimental_rerun()  # preferred public API
        return
    except Exception:
        pass
    try:
        from streamlit.runtime.scriptrunner.script_runner import RerunException  # internal API
        raise RerunException()
    except Exception:
        st.session_state["_safe_rerun_requested"] = True
        st.stop()
# handle session flag on app start
if st.session_state.get("_safe_rerun_requested"):
    st.session_state["_safe_rerun_requested"] = False  # clear flag
def compute_summary():  # aggregate KPIs and optional time-series
    summary = {
        'total_base': '-', 'total_ai': '-',}
'avg_queue_base': {'North': '-', 'East': '-', 'South': '-', 'West': '-'},
```

```
 'avg_queue_ai': {'North': '-', 'East': '-', 'South': '-', 'West': '-'},
      'ai_queue_reduction': '-', 'time_series': None
   }
if os.path.exists(PERF_BASE) and os.path.exists(PERF_AI):
      dfb = read_csv_safe(PERF_BASE)
      dfa = read_csv_safe(PERF_AI)
      if dfb is None or dfa is None:
         return summary
      total_base = compute_total_vehicles_from_df(dfb)
      total_ai = compute_total_vehicles_from_df(dfa)
      dirs = ['North', 'East', 'South', 'West']
      avg_base = {d: '-' for d in dirs}
      avg_ai = {d: '-' for d in dirs}
      lcols_b = [c.lower() for c in dfb.columns]
      if 'region' in lcols_b and 'avg_queue' in lcols_b:
         cmap = {c.lower(): c for c in dfb.columns}
         gb = dfb.groupby(cmap['region'])[cmap['avg_queue']].mean().to_dict()
         for d in dirs:
            v = gb.get(d, None)
            if v is not None and not pd.isna(v):
               avg_base[d] = fmt(v)
      else:
         cmap = {c.lower(): c for c in dfb.columns}
         if all(k in cmap for k in ['north', 'east', 'south', 'west']):
for d in dirs:
               try:
                  avg_base[d] = fmt(dfb[cmap[d.lower()]].mean())
               except Exception:
                  avg_base[d] = '-'
lcols_a = [c.lower() for c in dfa.columns]
      if 'region' in lcols_a and 'avg_queue' in lcols_a:
         cmap = {c.lower(): c for c in dfa.columns}
         ga = dfa.groupby(cmap['region'])[cmap['avg_queue']].mean().to_dict()
```

```
for d in dirs:
            v = ga.get(d, None)
            if v is not None and not pd.isna(v):
                avg_ai[d] = fmt(v)
else:
        cmap = {c.lower(): c for c in dfa.columns}
        if all(k in cmap for k in ['north', 'east', 'south', 'west']):
            for d in dirs:
                try:
                    avg_ai[d] = fmt(dfa[cmap[d.lower()]].mean())
                except Exception:
                    avg_ai[d] = '-'
    # reduction calculation
    try:
        s_base = sum([float(x) for x in avg_base.values() if x != '-'])
        s_ai = sum([float(x) for x in avg_ai.values() if x != '-'])
        red = round(100.0 * (s_base - s_ai) / s_base, 2) if (s_base not in (0, '-') and s_base !=
0) else '-'
    except Exception:
        red = '-'
    # optional time series extraction
    time_df = None
    for df_try in (dfb, dfa):
        cols_try = [c.lower() for c in df_try.columns]
 if ('step' in cols_try or 'time' in cols_try) and all(k in cols_try for k in ['north', 'east', 'south',
'west']):
            step_col = [c for c in df_try.columns if c.lower() in ('step', 'time')][0]
time_df = df_try[[step_col] + [c for c in df_try.columns if c.lower() in ('north', 'east', 'south',
'west')]].copy()
            time_df.columns = ['Step', 'North', 'East', 'South', 'West']
            break
    summary = {
 'total_base': total_base, 'total_ai': total_ai,
        'avg_queue_base': avg_base, 'avg_queue_ai': avg_ai,
        'ai_queue_reduction': red, 'time_series': time_df
    }
```

```
 return summary
def show_kpis_area(summary, placeholder):  # render KPI cards
    t_base = summary.get('total_base', '-')
    t_ai = summary.get('total_ai', '-')
    reduction = summary.get('ai_queue_reduction', '-')
    placeholder.markdown(f"""
        <div class="kpi-grid">
          <div class="kpi-card"><b>Total vehicles</b><br><span style="font-
size:22px">{t_base}</span><br><small>Baseline</small></div>
          <div class="kpi-card"><b>Total vehicles</b><br><span style="font-
size:22px">{t_ai}</span><br><small>AI</small></div>
          <div class="kpi-card"><b>AI queue reduction</b><br><span style="font-
size:22px">{reduction}%</span><br><small>Lower is better</small></div>
        </div>
        """, unsafe_allow_html=True)
def plot_all_charts(summary, placeholder):  # draw charts (bar, pie, donut, timeseries)
    dirs = ['North', 'East', 'South', 'West']
    base_vals = [float(summary['avg_queue_base'].get(d, 0)) if
summary['avg_queue_base'].get(d, '-') != '-' else 0 for d in dirs]
ai_vals = [float(summary['avg_queue_ai'].get(d, 0)) if summary['avg_queue_ai'].get(d, '-') !=
'-' else 0 for d in dirs]
fig_bar = go.Figure()
    fig_bar.add_trace(go.Bar(x=dirs, y=base_vals, name='Baseline', marker_color='#ff7a59'))
    fig_bar.add_trace(go.Bar(x=dirs, y=ai_vals, name='AI', marker_color='#ffd86b'))
    fig_bar.update_layout(barmode='group', title='Avg queue per direction',
template='plotly_white', height=380)
    placeholder.plotly_chart(fig_bar, use_container_width=True)
df_pie = pd.DataFrame({'Direction': dirs, 'Baseline': base_vals, 'AI': ai_vals})
    fig_p_base = px.pie(df_pie, names='Direction', values='Baseline', title='Baseline avg-
queue distribution')
    fig_p_ai = px.pie(df_pie, names='Direction', values='AI', title='AI avg-queue distribution')
    placeholder.plotly_chart(fig_p_base, use_container_width=True)
 placeholder.plotly_chart(fig_p_ai, use_container_width=True)
    try:
        red_val = float(summary.get('ai_queue_reduction', 0)) if
summary.get('ai_queue_reduction', '-') != '-' else 0.0
except Exception:
        red_val = 0.0
```

```
fig_donut = go.Figure(data=[go.Pie(values=[red_val, max(0, 100 - red_val)],
labels=['Reduced', 'Remaining'], hole=.6)])
    fig_donut.update_layout(title='AI Queue Reduction (%)', height=300)
    placeholder.plotly_chart(fig_donut, use_container_width=True)
    ts = summary.get('time_series', None)
    if ts is not None:
        try:
            ts['Total'] = ts[['North', 'East', 'South', 'West']].sum(axis=1)
            fig_line = go.Figure()
            fig_line.add_trace(go.Scatter(x=ts['Step'], y=ts['Total'], mode='lines+markers',
name='Total vehicles'))
            for d in ['North', 'East', 'South', 'West']:
                fig_line.add_trace(go.Scatter(x=ts['Step'], y=ts[d], mode='lines', name=d))
            fig_line.update_layout(title='Per-step vehicle counts', height=420,
template='plotly_white')
            placeholder.plotly_chart(fig_line, use_container_width=True)
        except Exception:
placeholder.info("Time-series not plotted due to unexpected format.")
# ---------- UI layout ----------
st.title("AI Powered Traffic Signal Processing")  # title
st.subheader("Simulation + Analytics Dashboard")  # subtitle
st.markdown("---")
# layout: left for file buttons + logs; right for controls
left_col, right_col = st.columns([1, 3])
with right_col:
    bc1, bc2, bc3, bc4 = st.columns([1, 1, 1, 0.6])  # control columns
    run_clicked = bc1.button("Run Simulation")  # run sim
    analyze_clicked = bc2.button("Analyze (show dashboard)")  # analyze
    show_kpi_clicked = bc3.button("Show KPI & Chart")  # show KPI
    reset_clicked = bc4.button("Reset State")  # reset
with left_col:
    st.header("Files & Logs")
    if st.button("Open: Baseline CSV"):
        df = read_csv_safe(PERF_BASE)
        if df is not None:
            st.dataframe(df)
```

```
st.download_button("Download baseline CSV", df.to_csv(index=False, encoding='utf-8'), file_name="performance_baseline.csv")
        else:
            st.warning("performance_baseline.csv not found or unreadable.")
    if st.button("Open: AI CSV"):
        df = read_csv_safe(PERF_AI)
        if df is not None:
            st.dataframe(df)
            st.download_button("Download ai CSV", df.to_csv(index=False, encoding='utf-8'), file_name="performance_ai.csv")
        else:
            st.warning("performance_ai.csv not found or unreadable.")
    if st.button("Open: AI Cycle CSV"):
        df = read_csv_safe(AI_CYCLE)
        if df is not None:
            st.dataframe(df)
st.download_button("Download ai_cycle_debug.csv", df.to_csv(index=False, encoding='utf-8'), file_name="ai_cycle_debug.csv")
        else:
            st.warning("ai_cycle_debug.csv not found or unreadable.")
    logs_box = st.empty()  # placeholder for logs
# quick reset
if reset_clicked:
    safe_rerun()
# run simulation (keeps previous synchronous streaming + log filtering)
if run_clicked:
    if not os.path.exists(SIM_SCRIPT_MAIN):
        st.error("adaptive_main.py not found in project folder.")
else:
        env = os.environ.copy()
 env["PYTHONIOENCODING"] = "utf-8"
        env["PYTHONUTF8"] = "1"
        try:
            proc = subprocess.Popen(["python", SIM_SCRIPT_MAIN], stdout=subprocess.PIPE, stderr=subprocess.STDOUT, text=True, env=env)
        except Exception as e:
            st.error(f"Failed to start simulation: {e}")
```

```
proc = None
    if proc:
        lines = []
        with st.spinner("Simulation running..."):
            while True:
                ln = proc.stdout.readline()
                if ln == "" and proc.poll() is not None:
                    break
                if ln:
                    low = ln.lower()
                    # filter YOLO noise and verbose library lines
                    if ("ultralytics" in low) or ("yolo disabled" in low) or ("no module named 'ultralytics'".lower() in low):
                        continue
lines.append(ln.rstrip())
                    logs_box.text("\n".join(lines[-300:]))
        proc.wait()
        logs_box.text("\n".join(lines[-300:] + ["Simulation finished."]))

        # optional compare script
        if os.path.exists(SIM_SCRIPT_COMPARE):
            try:
                pc = subprocess.Popen(["python", SIM_SCRIPT_COMPARE], stdout=subprocess.PIPE, stderr=subprocess.STDOUT, text=True, env=env)
                comp_lines = []
                with st.spinner("Running compare...")
while True:
                    cl = pc.stdout.readline()
                    if cl == "" and pc.poll() is not None:
if cl:
                        if ("ultralytics" in cl.lower()) or ("yolo disabled" in cl.lower()):
                            continue
                        comp_lines.append(cl.rstrip())
                        logs_box.text("\n".join(comp_lines[-300:]))
                pc.wait()
                logs_box.text("\n".join(lines[-300:] + ["Compare finished."]))
```

```
except Exception:
                st.warning("Compare script failed or produced no output.")
# Analyze => show KPI cards and charts
if analyze_clicked or show_kpi_clicked:
    summary = compute_summary()
    analytics = st.container()
    with analytics:
        st.markdown("## Analytics")
        show_kpis_area(summary, st)
        plot_all_charts(summary, st)
        summary_tbl = {
            'metric': ['Total vehicles (baseline)', 'Total vehicles (AI)', 'AI queue reduction (%)]
'value': [summary.get('total_base', '-'), summary.get('total_ai', '-'),
summary.get('ai_queue_reduction', '-')]
        }
        st.table(pd.DataFrame(summary_tbl))
st.markdown("---")
st.caption("Made with   AI Traffic Signal project.")  # footer
```

Chapter 6

RESULTS AND SNAPSHOTS

The displayed SUMO simulation image captures a highly congested four-way intersection operating under a traditional fixed-time traffic signal scheme. Long queues of vehicles have accumulated in all four approaches, with the West and South directions experiencing the most severe congestion.

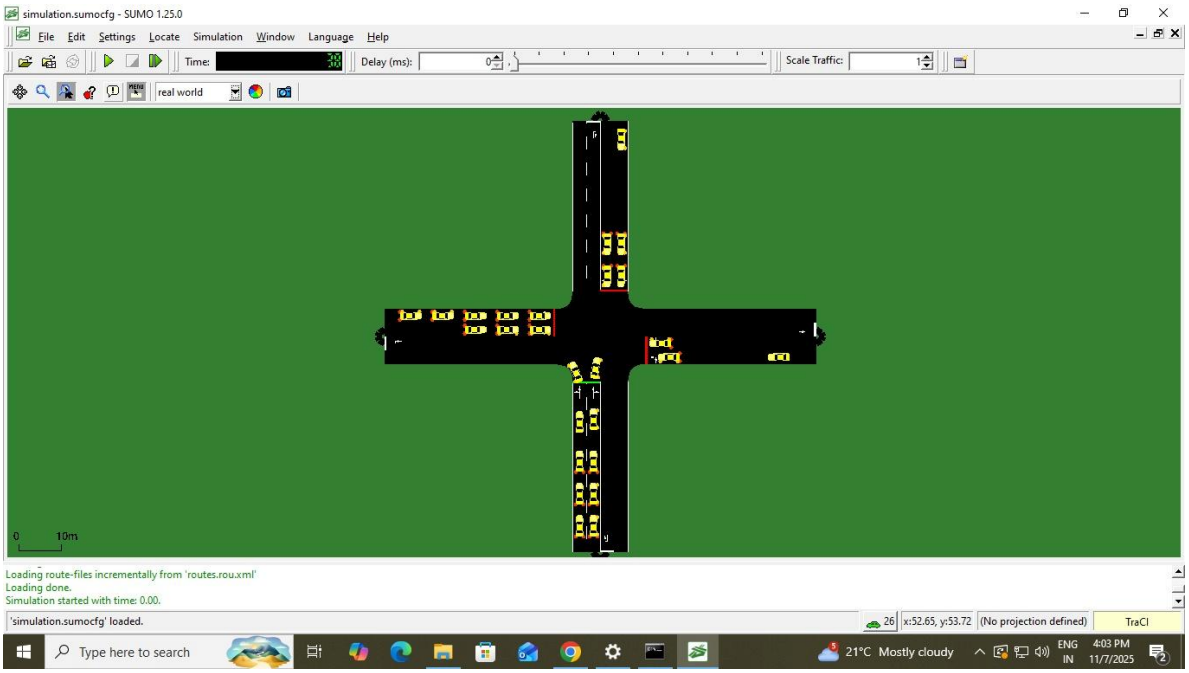


Fig. 6.1 SUMO Simulation Scene

The vehicles are lined up across multiple lanes, and the queues stretch deep into the upstream segments, indicating a persistent bottleneck effect. At this moment in the simulation, all traffic signals are red, forcing every vehicle to come to a complete halt. Since the fixed-time controller allocates equal green time to all directions irrespective of real-time traffic conditions, vehicles in the heavily loaded lanes are unable to clear efficiently. This mismatch between demand and signal timing leads to repeated stoppages and a rapid build-up of traffic density. The multi-lane setup with designated turning lanes further emphasizes the detailed modelling capabilities of SUMO's microscopic simulation environment. Each lane contains vehicles of uniform size and movement logic, demonstrating SUMO's realistic car-following and lane-change behaviour. The visible clusters of vehicles show typical effects of peak-hour traffic where fixed timing becomes insufficient. The layout also highlights the imbalance between inflow and outflow lanes with fewer vehicles are unnecessarily held at red signals, while busier lanes remain silent.

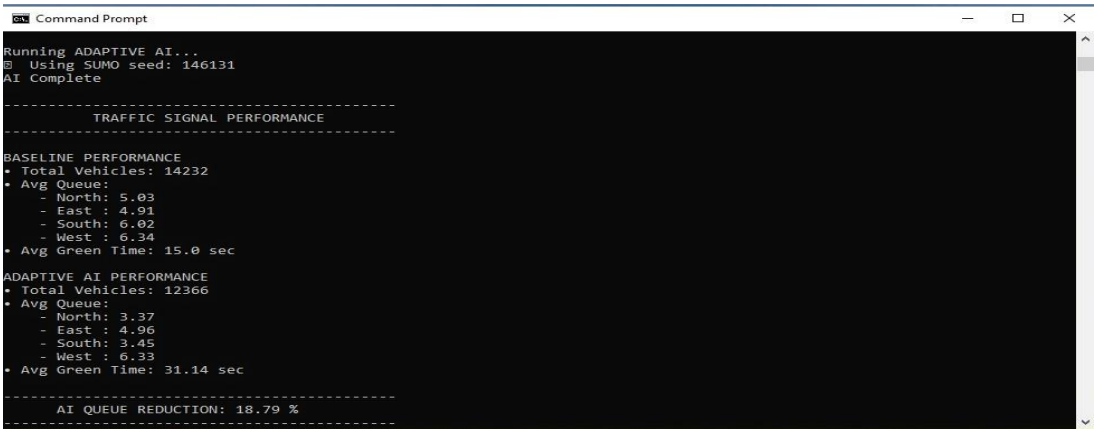


Fig. 6.2 SUMO Dashboard

The image presents the console output comparing the Baseline traffic signal system with the Adaptive AI system. The Baseline model displays higher average queue lengths across all directions, confirming that fixed-time controllers struggle with uneven traffic flows. In contrast, the Adaptive AI system shows a considerable reduction in queue length, particularly for the North and South directions. The console also reveals that the AI system dynamically adjusts green times, averaging 31.14 seconds compared to the constant 15-second green in the Baseline model. This demonstrates that the AI allocates more green time to busier lanes, improving overall efficiency. The final computed value an 18.79% reduction in queue length provides quantitative proof of the system's effectiveness. Overall, the console data validates that the adaptive method significantly enhances traffic performance, reducing delays and improving throughput.

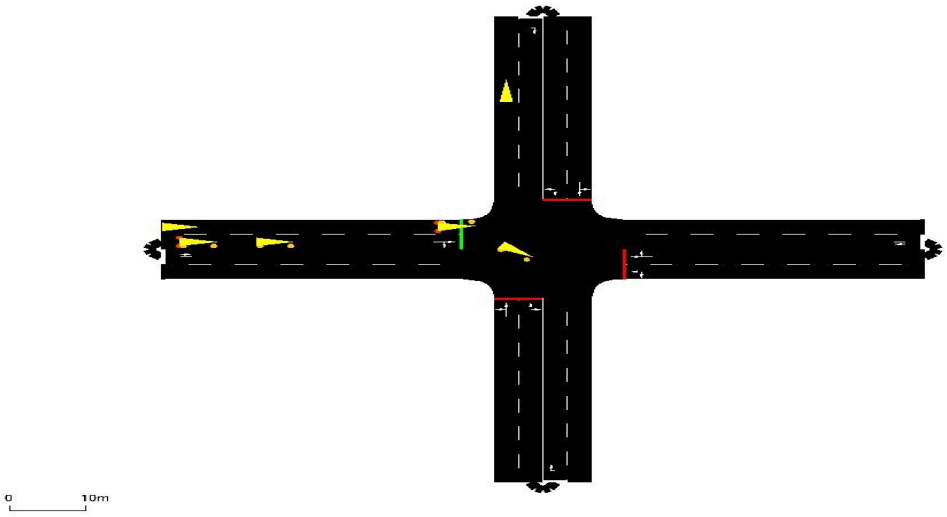


Fig. 6.3 SUMO Simulation Snapshot

The image displays a SUMO simulation snapshot of a four-way intersection where three approaches are currently facing a red signal, while one lane shows a green signal allowing vehicles to move. Vehicles are accurately positioned behind stop lines, forming visible queues typical of a signalized junction. Yellow arrows indicate possible turning movements, showcasing the complexity of the intersection layout. The multi-lane design, including straight and turning lanes, provides a realistic representation of urban traffic flow. This image captures the system in a phase where traffic begins to accumulate, highlighting how congestion builds during red-light intervals. Such scenarios demonstrate the importance of adaptive control, as fixed-timing cycles often fail to respond to fluctuating traffic loads. The visual highlights the early congestion that the AI aims to manage more effectively than traditional controllers.

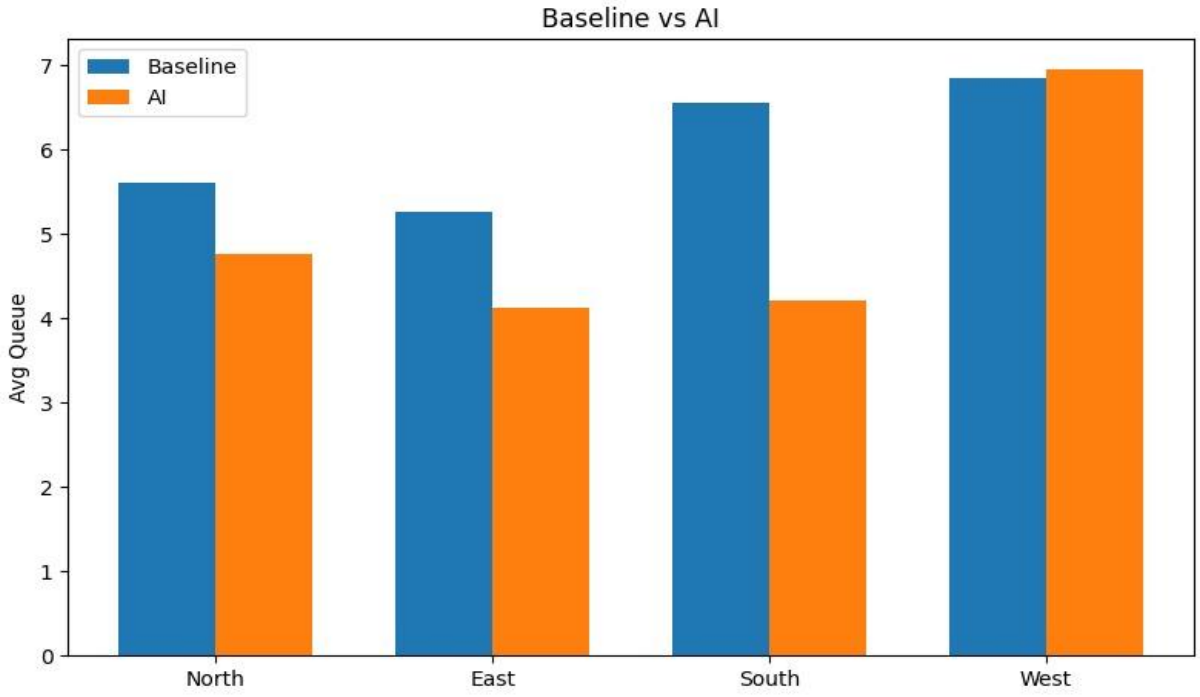
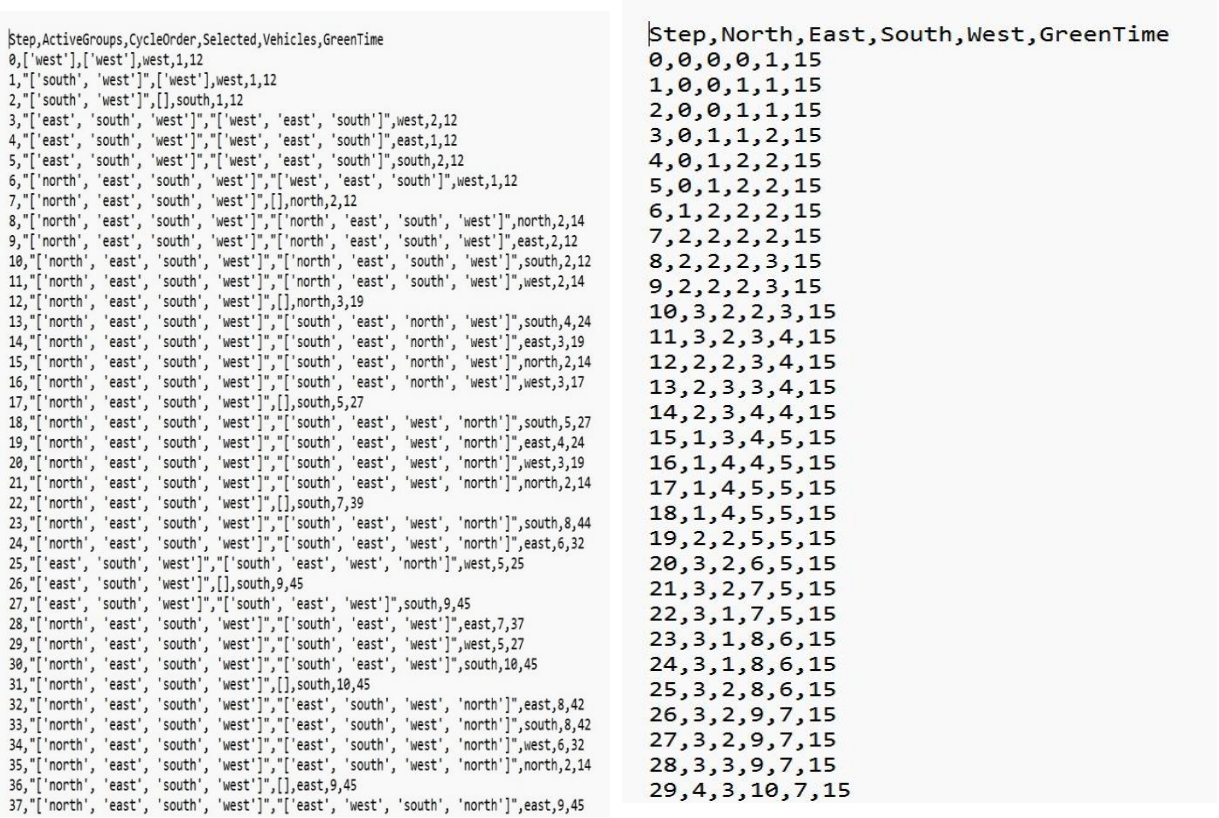


Fig. 6.4 Comparision Graph of Baseline vs AI

The image presents a comparative bar graph showing the average queue lengths for each approach North, East, South, and West under both the Baseline fixed-time signal and the AI adaptive signal system. The Baseline bars are consistently higher, indicating longer queues caused by rigid and unresponsive signal cycles. In contrast, the AI bars are noticeably lower for most approaches, especially North and South, where congestion typically builds fastest. This shows that the adaptive system is more effective in distributing green time based on real-time demand. The slight similarity in the West approach suggests naturally balanced traffic in

that direction. Overall, the graph clearly illustrates that the AI system reduces waiting time and smooths intersection flow, proving that real-time adaptive control significantly outperforms fixed-timing strategies. This visual evidence supports the improved performance of the proposed AI model.



```
Step,ActiveGroups,CycleOrder,Selected,Vehicles,GreenTime
0,['west'],['west'],west,1,12
1,"['south', 'west']",['west'],west,1,12
2,"['south', 'west']",[],south,1,12
3,"['east', 'south', 'west']","['west', 'east', 'south']",west,2,12
4,"['east', 'south', 'west']","['west', 'east', 'south']",east,1,12
5,"['east', 'south', 'west']","['west', 'east', 'south']",south,2,12
6,"['north', 'east', 'south', 'west']","['west', 'east', 'south']",west,1,12
7,"['north', 'east', 'south', 'west']",[],north,2,12
8,"['north', 'east', 'south', 'west']","['north', 'east', 'south', 'west']",north,2,14
9,"['north', 'east', 'south', 'west']","['north', 'east', 'south', 'west']",east,2,12
10,"['north', 'east', 'south', 'west']","['north', 'east', 'south', 'west']",south,2,12
11,"['north', 'east', 'south', 'west']","['north', 'east', 'south', 'west']",west,2,14
12,"['north', 'east', 'south', 'west']",[],north,3,19
13,"['north', 'east', 'south', 'west']","['south', 'east', 'north', 'west']",south,4,24
14,"['north', 'east', 'south', 'west']","['south', 'east', 'north', 'west']",east,3,19
15,"['north', 'east', 'south', 'west']","['south', 'east', 'north', 'west']",north,2,14
16,"['north', 'east', 'south', 'west']","['south', 'east', 'north', 'west']",west,3,17
17,"['north', 'east', 'south', 'west']",[],south,5,27
18,"['north', 'east', 'south', 'west']","['south', 'east', 'west', 'north']",south,5,27
19,"['north', 'east', 'south', 'west']","['south', 'east', 'west', 'north']",east,4,24
20,"['north', 'east', 'south', 'west']","['south', 'east', 'west', 'north']",west,3,19
21,"['north', 'east', 'south', 'west']","['south', 'east', 'west', 'north']",north,2,14
22,"['north', 'east', 'south', 'west']",[],south,7,39
23,"['north', 'east', 'south', 'west']","['south', 'east', 'west', 'north']",south,8,44
24,"['north', 'east', 'south', 'west']","['south', 'east', 'west', 'north']",east,6,32
25,"['east', 'south', 'west']","['south', 'east', 'west', 'north']",west,5,25
26,"['east', 'south', 'west']",[],south,9,45
27,"['east', 'south', 'west']","['south', 'east', 'west']",south,9,45
28,"['north', 'east', 'south', 'west']","['south', 'east', 'west']",east,7,37
29,"['north', 'east', 'south', 'west']","['south', 'east', 'west']",west,5,27
30,"['north', 'east', 'south', 'west']","['south', 'east', 'west']",south,10,45
31,"['north', 'east', 'south', 'west']",[],south,10,45
32,"['north', 'east', 'south', 'west']","['east', 'south', 'west', 'north']",east,8,42
33,"['north', 'east', 'south', 'west']","['east', 'south', 'west', 'north']",south,8,42
34,"['north', 'east', 'south', 'west']","['east', 'south', 'west', 'north']",west,6,32
35,"['north', 'east', 'south', 'west']","['east', 'south', 'west', 'north']",north,2,14
36,"['north', 'east', 'south', 'west']",[],east,9,45
37,"['north', 'east', 'south', 'west']","['east', 'west', 'south', 'north']",east,9,45
```

```
Step,North,East,South,West,GreenTime
0,0,0,0,1,15
1,0,0,1,1,15
2,0,0,1,1,15
3,0,1,1,2,15
4,0,1,2,2,15
5,0,1,2,2,15
6,1,2,2,2,15
7,2,2,2,2,15
8,2,2,2,3,15
9,2,2,2,3,15
10,3,2,2,3,15
11,3,2,3,4,15
12,2,2,3,4,15
13,2,3,3,4,15
14,2,3,4,4,15
15,1,3,4,5,15
16,1,4,4,5,15
17,1,4,5,5,15
18,1,4,5,5,15
19,2,2,5,5,15
20,3,2,6,5,15
21,3,2,7,5,15
22,3,1,7,5,15
23,3,1,8,6,15
24,3,1,8,6,15
25,3,2,8,6,15
26,3,2,9,7,15
27,3,2,9,7,15
28,3,3,9,7,15
29,4,3,10,7,15
```

Fig. 6.5 CSV file of Baseline and AI

This table provides a transparent view of the adaptive controller's behavior as it reacts to changing traffic conditions. For example, steps where higher vehicle counts appear (such as 7, 8, or 9 vehicles) lead to longer green durations like 39 or 45 seconds, demonstrating density-based proportional control. Conversely, steps with fewer vehicles receive shorter green times, reflecting efficient resource allocation and avoidance of unnecessary delays. The dynamic reordering of the cycle order reveals how the system continuously shifts priority depending on which approach becomes more congested. The presence of occasional empty vehicle values shows moments where some directions have cleared, allowing the controller to favor others. Overall, this step-by-step output verifies that the system is functioning as intended by consistently allocating green time where it is needed most, ensuring smoother flow and reduced queue formation.

Fig. 6.6 Frontend Dashboard

The dashboard interface shown above represents an AI-powered Traffic Signal Processing system designed to manage, simulate, and analyze traffic flow. The layout features a clean and minimal design with clearly defined functional sections. The header includes the project title "AI Powered Traffic Signal Processing – Simulation + Analytics Dashboard", emphasizing the system's purpose. The interface includes primary controls such as Run Simulation, Analyze(show dashboard), Show KPI & Chart, and Reset State, enabling users to conduct traffic simulations and view performance metrics with ease. A Files & Logs section provides quick access to essential datasets, including Baseline CSV, AI CSV, and AI Cycle CSV, which are used for comparative analysis between traditional and AI-optimized signal cycles. The overall design is user-friendly and visually intuitive, ensuring smooth navigation for simulation execution, data evaluation, and traffic signal performance monitoring. The footer notes that the project was created with passion, reinforcing the innovative, research-oriented intention of the system.

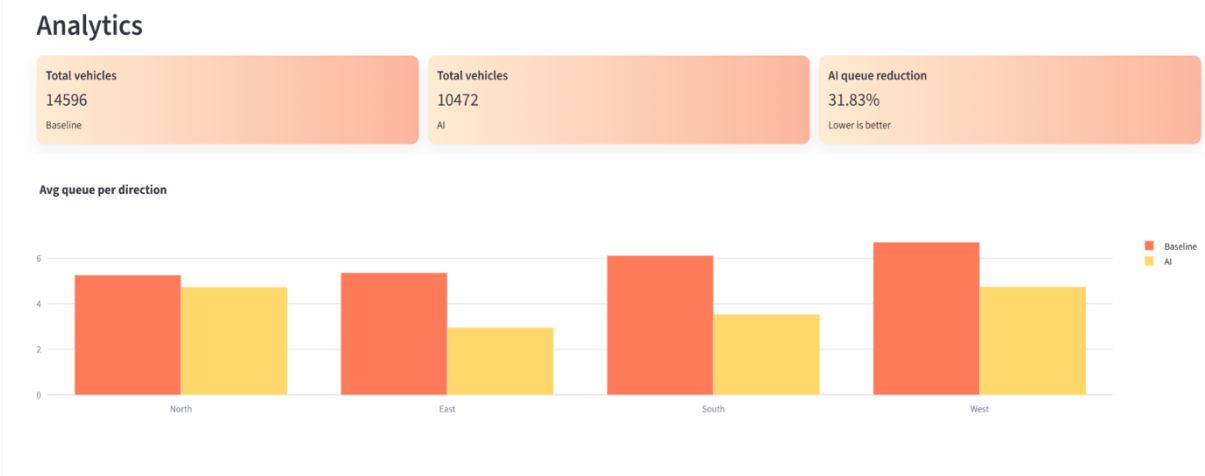


Fig. 6.7 Dashboard Information

Chapter 7

APPLICATIONS

The AI-powered adaptive traffic signal system developed in this project has a wide range of practical applications across modern transportation networks. With rapid urbanization and the increasing number of vehicles on the road, conventional fixed-time traffic controllers are no longer sufficient to handle dynamic and unpredictable traffic patterns. By integrating computer vision, deep learning, and real-time simulation, the proposed system provides an intelligent and data-driven approach to traffic management. Its ability to detect vehicles, estimate traffic density, and adjust signal timings autonomously makes it highly adaptable to various real-world environments. From smart cities and highways to emergency corridors and controlled campus zones, this system can significantly enhance mobility, reduce congestion, and improve overall road efficiency. The following applications highlight the diverse use-cases and the transformative impact of implementing adaptive traffic control technology in modern transportation infrastructure.

Urban Traffic Management

- The system optimizes signal timings at busy intersections by analyzing vehicle density.
- This reduces congestion and minimizes overall waiting time for road users.
- It leads to smoother and more efficient traffic movement across urban areas.

Smart City Infrastructure

- The project supports smart city initiatives by integrating AI, computer vision, and IoT technologies.
- It enables autonomous decision-making and adaptive traffic flow control.
- This creates a more intelligent, responsive, and future-ready transportation ecosystem.

Emergency Vehicle Prioritization

- The AI system can detect ambulances, fire trucks, and police vehicles instantly.
- Traffic signals can be adjusted to create a clear path by extending green lights.
- This significantly reduces emergency response time and improves public safety.

Pollution and Fuel Consumption Reduction

- By decreasing idle time at red lights, the system reduces unnecessary fuel burning.
- Smoother traffic flow lowers carbon emissions and air pollution.
- This contributes to cleaner and more sustainable urban environments.

Integration with Surveillance and Safety Systems

- The detection pipeline can be extended to monitor traffic violations in real time.
- It can identify behaviours like signal jumping, over speeding, or wrong-way driving.
- This enhances road safety and supports automated law enforcement.

Highway Ramp Metering and Toll Booth Management

- Adaptive timing controls vehicle inflow at highway entry ramps to prevent bottlenecks.
- Toll booth queues can also be managed dynamically to reduce waiting time.
- This ensures smooth transitions between local roads and fast-moving highway traffic.

Simulation Research and ITS Development

- SUMO and TraCI make this system ideal for testing new traffic algorithms.
- Researchers can experiment with AI models, routing strategies, and congestion control.
- This supports advancements in Intelligent Transportation Systems (ITS).

7.1 Contribution to Society and Environment

The AI-powered Adaptive Traffic Signal Control System developed in this project contributes significantly to both society and the environment by addressing some of the most critical challenges in modern urban transportation. Traffic congestion affects millions of people daily, leading to wasted time, increased stress, and reduced productivity. By intelligently adjusting signal timings based on real-time traffic density, the system ensures smoother vehicle movement and reduces waiting time at intersections. This directly improves the daily commuting experience, enhances road safety, and supports more efficient mobility within growing cities. The technology also benefits emergency services by enabling quicker passage for ambulances, fire trucks, and police vehicles through adaptive prioritization, potentially saving lives during critical situations.

From an environmental perspective, the system plays a vital role in promoting sustainable and eco-friendly transportation. Long idling times and stop-and-go traffic significantly increase fuel consumption and greenhouse gas emissions. By reducing unnecessary stops and clearing queues more efficiently, the adaptive controller decreases carbon dioxide output and fuel wastage. This contributes to cleaner air quality, reduces noise pollution, and supports broader environmental conservation goals. Moreover, by improving public transport prioritization, the system encourages the use of buses and shared mobility options, which further reduces the overall carbon footprint of urban transportation.

At a societal level, the implementation of intelligent traffic systems can support the evolution of smart cities, leading to better infrastructure planning and improved quality of life. The system generates valuable real-time traffic data that can help policymakers, urban planners, and government agencies design safer roads, optimize traffic corridors, and make informed decisions about future transport investments. In summary, this project not only advances technological innovation but also contributes meaningfully to societal well-being and environmental sustainability, making it a valuable step toward smarter, greener, and more efficient urban mobility.

Chapter 8

FUTURE ENHANCEMENTS

- Integration of Reinforcement Learning (RL) to enable the traffic controller to learn optimal signal policies through continuous interaction with the environment.

- RL-based systems can adapt to highly dynamic conditions and develop long-term strategies that reduce delays, queue lengths, and travel times.

- Implementation of multi-camera fusion to handle occlusions, shadows, and large vehicles by capturing the intersection from multiple angles.

- Integration of additional sensors such as LiDAR, radar, and IoT detectors to improve accuracy under challenging conditions like rain, fog, nighttime lighting, or camera vibration.

- Adoption of Vehicle-to-Infrastructure (V2I) communication, allowing connected vehicles to share real-time speed, position, and route data with the controller.

- Development of a multi-intersection, network-level adaptive system for city-wide coordination across multiple traffic signals.

- Creation of a cloud-based traffic dashboard for real-time monitoring, analytics, and centralized control over broad transportation networks.

- Introduction of priority mechanisms for emergency vehicles and public transport to ensure faster and safer movement through intersections.

- Enhancement of system reliability through hardware acceleration, edge computing, fail-safe mechanisms, and fallback modes during device or network failures.

- Integration with smart city platforms, predictive traffic models, and accident detection systems to evolve into a comprehensive intelligent transportation ecosystem.

Chapter 9

CONCLUSION

The development of the AI-powered Adaptive Traffic Signal Control System represents an important step toward modernizing conventional traffic management through intelligent automation. Traditional fixed-time traffic signals are unable to respond to dynamic and unpredictable traffic patterns, resulting in unnecessary delays, long queues, fuel wastage, and reduced road efficiency. This project overcomes these limitations by integrating computer vision, machine learning, and real-time simulation to create a system capable of understanding and reacting to actual traffic conditions. By using YOLOv8n for vehicle detection, the system accurately determines lane-wise traffic density, enabling the controller to allocate green time proportionally based on demand. SUMO simulation, along with TraCI-based Python controllers, allowed rigorous testing and validation of the adaptive model under realistic traffic scenarios. The results consistently showed reduced queue lengths, smoother flow, and significant performance improvements compared to fixed-time signals, demonstrating the effectiveness of intelligent control strategies.

Through systematic dataset preparation, annotation, model training, and network simulation, the project successfully showcases how AI can be integrated into transportation systems to improve mobility and efficiency. The performance analysis revealed that the adaptive controller dynamically adjusts to density fluctuations, alleviating congestion during peak conditions while maintaining balanced traffic distribution across all approaches. This adaptability is especially valuable in urban environments where traffic loads vary rapidly throughout the day. Additionally, the system's modular architecture makes it scalable and suitable for multi-intersection coordination, emergency vehicle prioritization, public transport support, and future smart-city deployments. Overall, the project highlights the transformative potential of AI in Intelligent Transportation Systems (ITS).

By combining deep learning, computer vision, and real-time simulation, the adaptive traffic signal achieves faster response times, reduced delays, environmental benefits through lower emissions, and improved traveller experience. The work serves as a foundation for future enhancements such as reinforcement learning-based optimization, integration with IoT sensors, cloud-based traffic coordination, and deployment on real-world intersections. In conclusion, this project demonstrates that AI-driven traffic control is not only feasible but highly impactful, paving the way for smarter, safer, and more efficient urban mobility systems.

REFERENCES

[1] Arafat, M. Y., Adnan, M. F., and Islam, M. F. (2023). AI-based affordable high-density traffic monitoring system. 2023 International Conference on Next-Generation Computing, IoT and Machine Learning (NCIM), 1–6.

[2] Gu, K., Hu, J., and Jia, W. (2023). Adaptive area-based traffic congestion control and management scheme based on fog computing. IEEE Transactions on Intelligent Transportation Systems, 24(1), 1359–1371.

[3] Uddin, M. I., Alamgir, M. S., Rahman, M. M., Bhuiyan, M. S., and Moral, M. A. (2021). AI traffic control system based on Deepstream and IoT using NVIDIA Jetson Nano. 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), 115–120.

[4] Abohashima, H., Gheith, M., and Eltawil, A. (2020). A proposed IoT based smart traffic lights control system within a V2X framework. Proceedings of NILES2020: 2nd Novel Intelligent and Leading Emerging Sciences Conference, 338–343.

[5] Li, J., Xie, D., Zhu, Q., and Wu, Z. (2023). Construction of intelligent transportation information management system based on artificial intelligence technology. 2023 2nd International Conference on Artificial Intelligence and Autonomous Robot Systems (AIARS), 1–6.

[6] Genitha, C. H., Danny, S. A., Ajibah, A. S. H., Aravint, S., and Sweety, A. A. V. (2023). AI based real-time traffic signal control system using machine learning. 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), 1613–1618.

[7] Subbiah, A. (2024). AI-enhanced IoT system for efficient traffic management: Leveraging black data in smart cities. 2024 International Conference on Intelligent Computing and Next Generation Networks (ICNGN), 1–7.

[8] Sirphy, S., and Revathi, S. T. (2023). Adaptive traffic control system using YOLO. 2023 International Conference on Computer Communication and Informatics (ICCCI), 1–6.

[9] Monica, C., Gottipati, S., Jyothi, B., Jahnavi, V., Chinnaiyan, R., Ramagiri, A., and Akther, S. A. (2023). Intelligent traffic monitoring, prioritizing and controlling model based on GPS. 2023 International Conference on Innovative Data Communication Technologies and Application (ICIDCA), 297–302.

[10] Singh, K., and Malik, N. (2022). CNN based approach for traffic sign recognition system. Asian Journal of Graduate Research, 11(1), 23–33.

[11] Shaygan, M., Meese, C., Li, W., Zhao, X. G., and Nejad, M. (2022). Traffic prediction using artificial intelligence: Review of recent advances and emerging opportunities. Transportation Research Part C: Emerging Technologies, 145, 103921.

[12] Sayed, S. A., Abdel-Hamid, Y., and Hefny, H. A. (2023). Artificial intelligence-based traffic flow prediction: A comprehensive review. Journal of Electrical Systems and Information Technology, 10(13).

[13] Assbeihat, J. M., and Rafi, N. (2021). Management of artificial intelligence traffic systems in smart cities. Academy of Strategic Management Journal, 20(S2), 1–12.

[14] Nampalli, R. C. (2021). Leveraging AI in urban traffic management: Addressing congestion and traffic flow with intelligent systems. Journal of Artificial Intelligence and Big Data, 1(1), 86–99.

[15] Mandal, V., Mussah, A. R., Jin, P., and Adu-Gyamfi, Y. (2020). Artificial intelligence-enabled traffic monitoring system. Sustainability, 12(21), 9177.

[16] Gilmore, J. F., and Elibiary, K. J. (1993). AI in advanced traffic management systems. AAAI Technical Report WS-93-04, 57–66.

[17] Ait Ouallane, A., Bahnasse, A., Bakalia, A., and Talea, M. (2022). Overview of road traffic management solutions based on IoT and AI. Procedia Computer Science, 198, 518–523.

[18] Zhang, X. (2025). Artificial Intelligence in Intelligent Traffic Signal Control. Proceedings of the 3rd International Conference on Software Engineering and Machine Learning.

[19] Fredianelli, L., Carpita, S., Bernardini, M., Del Pizzo, L. G., Brocchi, F., Bianco, F., and Licitra, G. (2022). Traffic flow detection using camera images and machine learning methods in ITS for noise map and action plan optimization. Sensors, 22(5), 1929.

[20] Qadri, S. S. S. M., Gökçe, M. A., and Öner, E. (2020). State-of-art review of traffic signal control methods: Challenges and opportunities. European Transport Research Review, 12(1), 55.

[21] Pan, T. (2023). Traffic Light Control with Reinforcement Learning. arXiv preprint arXiv:2308.14295.

[22] Ravish, R., and Swamy, S. R. (2021). Intelligent Traffic Management: A Review of Challenges, Solutions, and Future Perspectives. Transport and Telecommunication, 22(2), 163–182.

[23] Reza, S., Oliveira, H. S., Machado, J. J. M., and Tavares, J. M. R. S. (2021). Urban safety: An image-processing and deep-learning-based intelligent traffic management and control system. Sensors, 21(21), 7705.

[24] Kumar, N., and Raubal, M. (2021). Applications of deep learning in congestion detection, prediction and alleviation: A survey. arXiv preprint arXiv:2102.09759.

[25] Han, S., Chung, S.-J., and Gustafson, J. (2023). Congestion Control of Vehicle Traffic Networks by Learning Structural and Temporal Patterns. Proceedings of Machine Learning Research, 211, 1–12.

[26] Shrestha, B., Singh, B., Darlami, G., and Upadhayaya, N. (2025). AI Based Traffic Management System. International Journal of Educational Practices and Engineering (IJEPE), 2(1), 1–6.

[27] Saleem, M., Abbas, S., Ghazal, T. M., Khan, M. A., Sahawneh, N., and Ahmad, M. (2022). Smart cities: Fusion-based intelligent traffic congestion control system for vehicular networks using machine learning techniques. Egyptian Informatics Journal, 23(4), 417–426.

[28] Jurczenia, K., and Rak, J. (2022). A survey of vehicular network systems for road traffic management. IEEE Access, 10, 42365–42392.

[29] Khalil, R. A., Safelnasr, Z., Yemane, N., Kedir, M., Shafiqur Rahman, A., and Saeed, N. (2024). Advanced learning technologies for intelligent transportation systems: Prospects and challenges. IEEE Open Journal of Vehicular Technology, 5, 397–416.

[30] Raheem, S. B., Afolabi, A., and Ogunlade, J. (2025). The cause, effect and possible solution to traffic congestion on Nigeria road: A case study of Basorun-Akobo road, Oyo state.

[31] Hazarika, A., Choudhury, N., Nasralla, M. M., Khattak, S. B. A., and Rehman, I. U. (2024). Edge ML technique for smart traffic management in intelligent transportation systems. IEEE Access, 12, 25443–25455.

[32] Ismaeel, A. G., Mary, J., Chelliah, A., Logeshwaran, J., Mahmood, S. N., Alani, S., and Shather, A. H. (2023). Enhancing traffic intelligence in smart cities using sustainable deep radial function. Sustainability, 15(19), 14441.

[33] Almathami, Y. S. (2023). A proposal to a dynamic traffic detection system in Saudi Arabia: A sun-powered drones approach. 2023 IEEE 20th International Conference on Smart Communities (HONET).

[34] Rizwan, A., Karras, D. A., Dighriri, M., Kumar, J., Dixit, E., Jalali, A., and Mahmoud, A. (2022). Simulation of IoT-based vehicular ad hoc networks (VANETs) for smart traffic management systems. Wireless Communications and Mobile Computing, 2022.

[35] Shabbir, A., Cheema, A. N., Ullah, I., Almanjahie, I. M., and Alshahrani, F. (2024). Smart city traffic management: Acoustic-based vehicle detection using stacking-based ensemble deep learning approach. IEEE Access, 12, 35947–35959.

[36] Reddy, M. P. K., Vigneshwar, V. T., Shiva, M., and Revanth, B. (2025). AI-driven traffic management system using computer vision and ML. International Journal of Engineering Research and Science and Technology, 21(1), 83–88.

[37] Musa, A. A., Malami, S. I., Alanazi, F., Ounaies, W., Alshammari, M., and Haruna, S. I. (2023). Sustainable traffic management for smart cities using Internet-of-Things-oriented intelligent transportation systems (ITS): Challenges and recommendations.

[38] Gu, K., Hu, J., and Jia, W. (2023). Adaptive area-based traffic congestion control and management scheme based on fog computing. IEEE Transactions on Intelligent Transportation Systems, 24(1), 1359–1371.

[39] Uddin, M. I., Alamgir, M. S., Rahman, M. M., Bhuiyan, M. S., and Moral, M. A. (2021). AI traffic control system based on Deepstream and IoT using NVIDIA Jetson Nano. 2021 2nd International Conference on Robotics, Electrical and Signal Processing Techniques (ICREST), 115–120.

[40] Genitha, C. H., Danny, S. A., Ajibah, A. S. H., Aravint, S., and Sweety, A. A. V. (2023). AI based real-time traffic signal control system using machine learning. 2023 4th International Conference on Electronics and Sustainable Communication Systems (ICESC), 1613–1618.

APPENDIX

Acceptance Letter to present Literature Survey paper in 2nd International Symposium on Collaborative Informatics (2nd ISCI 2025)

2nd INTERNATIONAL SYMPOSIUM ON COLLABORATIVE INFORMATICS (2nd ISCI 2025)

17-19 December 2025 Venue: University BDT College of Engineering, Davanagere, INDIA

Conveners:
Dr. C M Ravi Kumar and Dr M H Divakar (India)
Prof. Toshinobu YAMAGUCHI and Prof Tomonori SAITA (Japan)

November 16, 2025

Mr/Ms R Alekhya
Undergraduate Student
Department of Artificial Intelligence and Data Science
K S School of Engineering and Management
Bangalore, India
Email: alekhyar2005@gmail.com

Paper No: 2nd ISCI2025-171

Dear Sir/Madam,

Thank you for your interest in the **2nd International Symposium on Collaborative Informatics (2nd ISCI 2025)** to be held **at University BDT College of Engineering, Davanagere during 17-19 December 2025**.

We are pleased to inform you that your paper **AI Powered Traffic Signal Processing** (Authors: Harshitha V, M Joshna, Mokshitha S Thota, R Alekhya and P S Geetha) has been accepted for **oral presentation** at 2nd ISCI 2025.

The guidelines for preparing the final manuscript, in msword format, are enclosed. *(Font: Verdana; Title of paper 16 (bold); author name 10 (bold), affiliation 10; titles 10 (bold); text 10; single column format; single spacing; maximum length: **12** pages).*

Please send us (ubdtisci2025@gmail.com with a copy to katta@nitk.edu.in) the manuscript as soon as possible, and not later than **November 30, 2025.** The full paper also will be reviewed by our technical committee. For more details, please refer to the information on the Symposium website. https://basukumbar.wixsite.com/ubdtcesymposium

The bank details for remitting the registration fee will be informed when you submit the final manuscript. The detailed program schedule will be sent to you in due course.

We are looking forward to meeting you in Davanagere.

With best regards,

Sincerely Yours

C M Ravikumar
Associate Professor
Department of Civil Engineering
University BDT College of Engineering
Davanagere-577004, INDIA
Email: cmravibdt@gmail.com
Phone: +91 9845001540

M H Divakar
Coordinator
Department of Robotics & Artificial Intelligence
University BDT College of Engineering
Davanagere-577004, INDIA
Email: divakarmh@gmail.com
Phone: +91 98442 49074