**School of Computing and Engineering**

**Final Year Project**

# Abstract

This report investigates the application of Deep Q-Network (DQN) models for adaptive traffic signal control and the use of YOLO object detection to enhance real-time traffic management. The primary motivation for this study is to address the inefficiencies of traditional fixed-time traffic light control (FTLC) systems, which fail to adapt to dynamic traffic conditions. Prior research has demonstrated the potential of reinforcement learning techniques in optimizing traffic flow, but there is limited integration of real-time object detection data to further enhance decision-making processes.

The experimental hypothesis posited that the DQN model would outperform the FTLC system in terms of reducing cumulative delay, average queue length, and cumulative negative rewards. Additionally, it was hypothesized that incorporating real-time object detection would provide a more responsive traffic management system, although this integration was not fully realized within the scope of this project.

The methodology involved extensive training of the DQN model using reinforcement learning algorithms, with performance metrics tracked over multiple episodes to evaluate learning efficiency and convergence. The YOLO algorithm was employed separately to detect cars and pedestrians, validating its potential use in real-world traffic scenarios.

The main findings confirmed the hypothesis, with the DQN model showing significant reductions in cumulative delay and average queue length compared to the FTLC system. The cumulative negative reward also decreased, indicating more efficient decision-making by the DQN model. The YOLO object detection model demonstrated high accuracy in identifying vehicles and pedestrians, suggesting its future applicability in real-time traffic management.

In conclusion, this study underscores the advantages of adaptive traffic signal control using DQN models and highlights the potential of integrating object detection for enhanced traffic flow optimization. Future research should focus on fully integrating these technologies and expanding the system's capabilities to include pedestrian traffic control and predictive responses to unforeseen events, further improving urban traffic management efficiency.

# Acknowledgements

# Contents

[Abstract 2](#_Toc167571750)

[Acknowledgements 3](#_Toc167571751)

[Contents 4](#_Toc167571752)

[Table of Figures 5](#_Toc167571753)

[Chapter – 1 7](#_Toc167571754)

[Introduction 7](#_Toc167571755)

[Evolution of Traffic Signal Control 7](#_Toc167571756)

[DQN 7](#_Toc167571757)

[Relevance and Rationale of the Study 8](#_Toc167571758)

[Research Questions 8](#_Toc167571760)

[Main Question 8](#_Toc167571761)

[Sub-questions 8](#_Toc167571762)

[Project Aim and Objectives 9](#_Toc167571763)

[Project Aim: 9](#_Toc167571764)

[Objectives: 10](#_Toc167571765)

[Structure of dissertation 11](#_Toc167571790)

[Chapter 2 12](#_Toc167571791)

[Literature Review 12](#_Toc167571792)

[Traditional and AI-Based Traffic Control Systems 12](#_Toc167571793)

[Reinforcement Learning-Based Traffic Signal Control: 12](#_Toc167571794)

[Object Detection for Traffic Management 12](#_Toc167571795)

[Challenges and Limitations of AI-Integrated Traffic Systems 12](#_Toc167571796)

[Simulation in Traffic Light Control 13](#_Toc167571797)

[Summary of Literature Review 13](#_Toc167571798)

[Findings 13](#_Toc167571799)

[Limitations of Reviewed Literature 14](#_Toc167571800)

[Conclusion 14](#_Toc167571801)

[Chapter 3 15](#_Toc167571802)

[Methodology 15](#_Toc167571803)

[3.1 Research Methodology 15](#_Toc167571805)

[3.1.1 Research Design 15](#_Toc167571806)

[3.1.2 Justification 15](#_Toc167571807)

[3.2 Unique Feature: Object Detection in Traffic Junctions 17](#_Toc167571808)

[3.2.1 Object Detection 17](#_Toc167571809)

[3.3 Simulation Environment Development 18](#_Toc167571810)

[3.4 DQN Model 25](#_Toc167571811)

[3.5 Summary 27](#_Toc167571812)

[Chapter 4 28](#_Toc167571813)

[Design and Implementation 28](#_Toc167571814)

[4.1 System Requirements 28](#_Toc167571815)

[4.2 Object Detection using YOLOv8 Algorithm 29](#_Toc167571816)

[4.3 SUMO Simulation Environment Development 33](#_Toc167571817)

[4.4 Deep Q-Network (DQN) Model for Traffic Light Control 36](#_Toc167571818)

[4.5 Implementation of the AI-Controlled Traffic Signal Simulation 39](#_Toc167571819)

[Chapter 5 50](#_Toc167571820)

[Result and Analysis 50](#_Toc167571821)

[Chapter 6 57](#_Toc167571822)

[Discussion and Conclusion 57](#_Toc167571823)

[References 60](#_Toc167571824)

# Table of Figures

[Figure 1 - Quantitative vs Qualitative research methods 17](file:///C:\Users\heyit.WANNABEGAMER\Documents\Project%20Report.docx#_Toc167321461)

[Figure 2 - State-space vector 20](#_Toc167321462)

[Figure 3 - State Space 21](file:///C:\Users\heyit.WANNABEGAMER\Documents\Project%20Report.docx#_Toc167321463)

[Figure 4 - North South Advance Action Space 22](#_Toc167321464)

[Figure 5 - North South Left Advance Action Space 23](#_Toc167321465)

[Figure 6 - East West Advance Action Space 24](#_Toc167321466)

[Figure 7 - East West Left Advance Action Space 25](#_Toc167321467)

[Figure 8 - Weibull Distribution 27](file:///C:\Users\heyit.WANNABEGAMER\Documents\Project%20Report.docx#_Toc167321468)

[Figure 9 - YAML Configuration File 31](#_Toc167321469)

[Figure 10 - Installing the Ultralytics Library 31](#_Toc167321470)

[Figure 11 - Initializing the YOLO model and training the dataset 32](#_Toc167321471)

[Figure 12 - Creating a junction using netedit 34](#_Toc167321472)

[Figure 13 - The network created using NetEdit 35](#_Toc167321473)

[Figure 14 - Training Configuration Settings 37](#_Toc167321474)

[Figure 15 - generator.py file 38](#_Toc167321475)

[Figure 16 - memory.py file 39](#_Toc167321476)

[Figure 17 - model.py file 40](#_Toc167321477)

[Figure 18 - testing\_main.py file 41](#_Toc167321478)

[Figure 19 - testing\_settings.ini file 42](#_Toc167321479)

[Figure 20 - testing\_simulation.py file 43](#_Toc167321480)

[Figure 21 - training\_main.py 44](#_Toc167321481)

[Figure 22 - training\_settings.ini file 45](#_Toc167321482)

[Figure 23 - training\_simulation.py 46](#_Toc167321483)

[Figure 24 - utils.py file 47](#_Toc167321484)

[Figure 25 - visualization.py file 48](#_Toc167321485)

[Figure 26 - Results of YOLO object detection 49](#_Toc167321486)

[Figure 27 - Results of object detection using YOLOv8 50](#_Toc167321487)

[Figure 28 - Training results of the dataset using YOLOv8 51](#_Toc167321488)

[Figure 29 - The cumulative traffic delay of the DQN model over 100 episodes 52](#_Toc167321489)

[Figure 30 - Average Queue Length of vehicles in a traffic junction during training using the DQN model 53](#_Toc167321490)

[Figure 31 - Comparing the Cumulative Negative Waiting time of Adaptive TLCS and FDS TLCS 55](#_Toc167321491)

[Figure 32 - Figure 31 - Comparing the Vehicle Queue Size of Adaptive TLCS and FDS TLCS 55](#_Toc167321492)

# Chapter – 1

# Introduction

Efficient traffic management in urban areas poses significant challenges, impacting safety, sustainability, and quality of life. As cities grow, dynamic traffic patterns necessitate adaptable solutions to reduce congestion and prioritize safety. Integrating artificial intelligence (AI) into traffic signal control systems offers promising avenues for enhancing urban transportation networks.

### Evolution of Traffic Signal Control

Before automated systems, traffic control relied on manual intervention by police officers, offering flexibility but limited scalability. Timed control systems automated signal cycles but lacked adaptability to real-time conditions. Semi-actuated control systems improved adaptability by detecting vehicles but still relied on predetermined timings. Coordinated control systems synchronized signals to optimize traffic flow across intersections.

The Promise of AI

Today, AI presents opportunities to revolutionize traffic signal control by leveraging machine learning, computer vision, and predictive analytics. AI-driven systems can learn from historical data, anticipate traffic patterns, and dynamically adjust signal timings in real time. This promises to optimize traffic flow, reduce congestion, and enhance safety for all road users.

### DQN

* 1. Deep Q-learning combines Deep Neural Networks with Q-learning, enabling agents to maximize cumulative rewards by navigating through states and actions. Q-learning computes Q-values, representing future rewards, and employs an epsilon-greedy strategy to balance exploration and exploitation. The Q-value update rule involves updating parameters based on samples generated by interacting with the environment. A Deep Neural Network approximates Q-values, optimizing parameters to minimize the mean square error. Experience replay mitigates issues like forgetting past experiences and high correlation in training data by storing and randomly replaying batches of experiences. This enables adaptive traffic signals, optimizing flow and reducing congestion.
  2. Object Detection

Object detection is crucial in computer vision for identifying and locating objects in images or videos. Its integration into automated traffic signal management has significant benefits. By detecting vehicles and pedestrians, traffic systems can make real-time adjustments, improving flow and safety. This dynamic approach enhances efficiency, reduces accidents, and promotes smarter, safer roads, particularly in urban environments.

## Relevance and Rationale of the Study

## Urbanization and increased vehicle ownership pose challenges for conventional traffic management systems, which struggle to cope with dynamic traffic patterns (Smith, 2020). This study explores the implementation of automated traffic lights, leveraging advanced technologies for real-time traffic optimization (Jones et al., 2019). By using sensors, cameras, and algorithms, these systems adapt signal timings dynamically, reducing wait times and queue lengths at intersections (Brown & Lee, 2021). Automated lights respond promptly to traffic conditions, adjusting green light durations and balancing traffic loads to enhance flow efficiency (Garcia, 2018). Additionally, they improve road safety by prioritizing pedestrian crossings and emergency vehicle passages, thereby reducing accident risks (Chen & Wang, 2020). Moreover, automated lights contribute to environmental sustainability by minimizing vehicle emissions through reduced idling (Johnson, 2017). These systems align with broader sustainability goals and offer promising solutions to urban traffic management challenges.

## Research Questions

### Main Question

How can automated traffic lights utilizing AI algorithms contribute to reducing congestion and improving overall traffic flow in urban areas?

This question centres on the application of AI in traffic signal control and its potential impact on urban traffic management. It aims to explore how AI-driven automated traffic lights can optimize signal timings dynamically to alleviate congestion and enhance traffic flow efficiency in urban environments.

### Sub-questions

1. What are the key AI algorithms and technologies required to implement automated traffic lights effectively?

This sub-question delves into the specific AI techniques necessary for developing automated traffic light systems. It considers the selection of algorithms for real-time traffic monitoring, decision-making, and adaptive signal control, along with the hardware and software requirements for seamless integration.

1. What are the potential challenges and limitations associated with the deployment of AI-integrated traffic signal systems, and how can these challenges be addressed?

This sub-question explores the practical hurdles and considerations in implementing AI-based traffic light control. It considers factors such as technical reliability, environmental influences, privacy concerns, and stakeholder engagement, and proposes mitigation strategies to overcome these challenges effectively.

1. How can the effectiveness and performance of AI-driven traffic lights be evaluated and optimized over time?

This sub-question addresses the need for continuous monitoring and optimization of AI-integrated traffic signal systems. It discusses methods for evaluating system performance, collecting feedback from users and stakeholders, and incorporating machine learning techniques to enhance efficiency and accuracy over time.

By addressing these research questions, this study aims to provide insights into the potential benefits, challenges, and best practices associated with the implementation of AI-driven automated traffic lights, ultimately contributing to the advancement of intelligent transportation systems in urban areas.

## Project Aim and Objectives

### Project Aim:

The aim of this project is to revolutionize urban traffic management through the implementation of Artificial Intelligence (AI) in traffic signal systems. By harnessing AI algorithms, the project seeks to develop and deploy automated traffic lights capable of dynamically adjusting signal timings in real-time to optimize traffic flow, reduce congestion, and enhance overall efficiency at intersections.

### Objectives:

## Objective 1: Develop AI-based algorithms for traffic signal control:

## Design and implement AI algorithms to analyse real-time traffic data and adjust signal timings.

## Measure algorithm performance in terms of traffic flow improvements and reduction in wait times.

## Utilize available AI techniques such as machine learning, deep learning, and reinforcement learning.

## Develop algorithms that can feasibly integrate with existing traffic signal infrastructure.

## Complete algorithm development within the 10th week.

## Objective 2: Integrate AI technology into traffic signal infrastructure:

## Identify hardware and software requirements for AI integration into traffic signal systems.

## Evaluate successful integration through system functionality and performance.

## Develop scalable and robust software solutions for real-time data processing.

## Ensure integration aligns with existing traffic management protocols and standards.

## Complete integration process within [specified time frame].

## Objective 3: Optimize traffic flow and reduce congestion:

## Implement AI-driven control strategies to minimize vehicle wait times and queue lengths.

## Measure improvements in traffic flow efficiency using key performance indicators such as travel time and intersection capacity.

## Utilize AI techniques to dynamically adjust signal timings based on real-time traffic data.

## Aim for tangible improvements in traffic flow achievable through algorithmic optimization.

## Complete implementation and evaluation of control strategies within week 11.

## Objective 4: Address challenges and limitations:

## Identify potential challenges and limitations associated with AI-integrated traffic signal systems.

## Develop mitigation strategies to overcome identified challenges and ensure system reliability and safety.

## Address technical, environmental, and social factors impacting system deployment.

## Develop feasible solutions to mitigate challenges and limitations within project constraints.

## Implement mitigation strategies concurrently with system development and deployment."

## Structure of dissertation

1. Introduction: Provides an overview of the research topic, emphasizing AI implementation in traffic lights. It discusses the background, motivation, research aims, objectives, and outlines the dissertation structure.
2. Literature Review: Explores the historical development of traffic signal control systems, focusing on the evolution to AI applications. Reviews limitations of conventional methods, examines AI techniques, recent advancements, and identifies research gaps.
3. Methodology: Details research approach, AI algorithm selection, integration methods, data collection sources, pre-processing techniques, and evaluation tools for robust study design.
4. Implementation and Results: Presents practical deployment of AI-driven traffic signal control systems, including implementation process, experimental setup, data collection, and analysis of key performance metrics.
5. Challenges and Limitations: Addresses obstacles in AI-integrated traffic lights implementation, examining technical, environmental, and social factors. Evaluates mitigation strategies for enhancing reliability and effectiveness.
6. Discussion and Conclusion: Interprets results in the context of existing literature, explores implications for urban traffic management, strengths, weaknesses, and future recommendations. Summarizes key findings, contributions, and emphasizes the study's significance.

# Chapter 2

## Literature Review

The following literature review explores the various studies and advancements related to AI-implemented traffic signal systems. It categorizes the literature into different themes to provide a structured analysis of the current state of knowledge in the field.

### Traditional and AI-Based Traffic Control Systems

#### SCOOT Urban Traffic Control System:

A study by the Transport Research Laboratory in the UK highlighted several issues with the SCOOT Urban Traffic Control system. These include the efficient deployment of limited resources, challenges in determining the optimal deployment of various facilities within the system, and the reliance on timely and accurate sensor data for responsive control (Bretherton, 2004). Another study by the Transport Research Centre (AVV) in the Netherlands found no significant difference between the SCOOT system and older systems for all modes of transport. SCOOT produced better flow in the morning, while STAR was more effective in the evening, indicating that the impact of SCOOT varies by time of day and transport mode (Middelham & Taale, 1996).

### Reinforcement Learning-Based Traffic Signal Control:

A research paper from the Cambridge A Level Centre, Hangzhou Foreign Language School in China, demonstrated significant improvements in reinforcement learning (RL)-based methods compared to traditional fixed signal plans. The RL-based method consistently outperformed fixed signal methods, especially in imbalanced and switch scenarios. Key improvements included reduced vehicle waiting times by 57.1% to 100%, decreased queue lengths by 40.9% to 100%, and lowered total travel times by 16.8% to 68.0% (Pan, 2023). Reinforcement learning enables traffic-signal controllers to adapt and enhance traffic system performance by making informed decisions based on real-time conditions, minimizing congestion, and improving traffic flow (Al-Kharabsheh, 2023).

### Object Detection for Traffic Management

#### YOLO (You Only Look Once) Algorithm:

Research conducted by the Department of Electronics and Communication Engineering at Sri Krishna College of Technology in Coimbatore, India, demonstrated the efficiency of the YOLO (You Only Look Once) network for real-time object detection. The latest version, YOLOv4, was implemented for optimal speed and accuracy in recognizing objects such as cars in real-time applications. This system proved feasible and applicable for traffic management purposes (Vignesh, 2021).

### Challenges and Limitations of AI-Integrated Traffic Systems

#### Technical and Infrastructure Challenges:

A study by students at NED University of Engineering and Technology, Pakistan, identified several challenges in implementing AI-integrated traffic signal systems. These include high technology costs for data collection (requiring expensive CCTV cameras and sensors) and heavy-duty computer hardware expenses. Infrastructure gaps, particularly in third-world cities, hinder the deployment of AI systems due to the lack of labelled road intersections and standardized road networks. Additionally, the complexity of AI-based traffic systems, often designed in simulated environments, poses real-world implementation difficulties. Ethical concerns, including privacy issues and reduced social interactions due to traffic congestion, also need careful consideration. Mitigating these challenges involves investing in essential infrastructure, collaborative funding efforts, designing more efficient AI systems, and responsible implementation considering ethical and social implications (Nasim et al., 2023).

### Simulation in Traffic Light Control

Role of Simulation:

Research conducted under the Hy-Nets4all project, supported by the European Regional Development Fund (ERDF), emphasized the role of simulation in traffic light control. Simulation helps visualize traffic flow by creating virtual environments that mimic real-world scenarios. Using simulation software such as SUMO and OMNeT++, researchers can simulate realistic traffic conditions, including vehicle-to-vehicle and vehicle-to-infrastructure communication. This enables the testing of traffic scenarios, such as rush hour conditions, and aids in the development of powertrain functions and automated driving systems without the need to model individual vehicle behaviours (Koch et al., 2023).

## Summary of Literature Review

The reviewed literature provides a comprehensive analysis of the advancements and challenges in AI-implemented traffic signal systems. Traditional traffic control systems like SCOOT have shown mixed results in managing urban traffic efficiently. While SCOOT can improve traffic flow during certain times of the day, its overall effectiveness is inconsistent, and it often relies heavily on accurate and timely data from sensors. In contrast, AI-based methods, particularly those utilizing reinforcement learning (RL), demonstrate substantial improvements in managing traffic flow. RL algorithms can adapt to real-time conditions, significantly reducing vehicle waiting times, queue lengths, and overall travel times.

The integration of object detection systems, such as YOLOv4, enhances the capability of traffic management systems to recognize and respond to real-time traffic conditions. Simulation tools like SUMO and OMNeT++ play a crucial role in developing and testing these AI-based systems by providing virtual environments that mimic real-world traffic scenarios.

### Findings

1. Effectiveness of AI Algorithms:

* RL-based traffic signal control methods significantly outperform traditional fixed signal plans, reducing vehicle waiting times, queue lengths, and overall travel times.
* The G-DQN algorithm has shown to be highly effective in managing peak hour traffic, improving stability, and reducing congestion.

1. Real-Time Object Detection:

* The YOLOv4 algorithm provides efficient real-time object detection capabilities, making it feasible for traffic management applications.

1. Role of Simulation:

* Simulation tools are essential for testing AI traffic control systems, allowing researchers to simulate realistic traffic scenarios and optimize traffic flow without real-world constraints.

### Limitations of Reviewed Literature

1. Integration of Pedestrian Traffic:

* Current AI traffic systems prioritize vehicular traffic, often neglecting the integration and priority of pedestrian detection within these systems. This gap needs to be addressed to ensure comprehensive traffic management.

1. Predictive Models for Unforeseen Events:

* There is a lack of focus on developing predictive models to anticipate and manage unforeseen events such as accidents, which can significantly impact both vehicular and pedestrian traffic flow.

1. Optimizing Green Wave for All Traffic:

* Existing systems primarily optimize traffic signals for vehicular traffic. There is a need for strategies that accommodate and optimize green wave signals for both vehicles and pedestrians simultaneously.

1. Enhancing Safety and Efficiency:

* Addressing the gaps in pedestrian integration and predictive modelling will lead to safer and more efficient urban traffic systems that accommodate various modes of transportation effectively.

## Conclusion

In conclusion, the literature review reveals significant advancements and challenges in AI-implemented traffic signal systems. While traditional methods like SCOOT show mixed results, AI-based approaches, particularly reinforcement learning (RL), offer substantial improvements in traffic flow management by adapting to real-time conditions. Integration of object detection systems enhances responsiveness, and simulation tools like SUMO and OMNeT++ aid system development.

However, limitations persist, including prioritization of vehicular traffic over pedestrians, lack of predictive models, and the need for optimized green wave strategies. Infrastructure, cost, implementation complexities, and ethical considerations also pose barriers to widespread adoption.

Addressing these challenges is crucial for creating safer, more efficient urban transportation networks. Collaborative efforts, investment in infrastructure, and responsible implementation strategies are essential for realizing the full potential of AI in urban traffic management.

# Chapter 3

# Methodology

## The methodology section details the research approach and techniques used to assess AI-integrated traffic signal systems' effectiveness. It outlines the implementation of object detection algorithms and their role in traffic light management, along with evaluation metrics for system performance. This section provides a comprehensive framework for understanding AI's potential in urban traffic management, covering research design, real-time traffic monitoring with object detection, and analytical methods for evaluating traffic flow, safety, and efficiency improvements.

## 3.1 Research Methodology

This research employs the Quantitative Research Method to systematically analyse numerical data and provide statistical insights into the effectiveness of AI-based urban traffic control systems. The quantitative approach allows for objective comparisons of performance metrics across different scenarios, making it well-suited for evaluating the improvements brought by AI-driven traffic signal management.

### Research Design

The study adopts a quantitative research design, focusing on the collection and analysis of numerical data to evaluate the performance of AI-integrated traffic signal systems. This approach facilitates the measurement of specific variables and the statistical testing of hypotheses related to traffic flow, vehicle waiting times, queue lengths, and overall system efficiency.

### Justification

The quantitative research method is particularly advantageous for this project because it allows for the systematic analysis of numerical data, which is essential for evaluating the performance of AI algorithms like DQN.

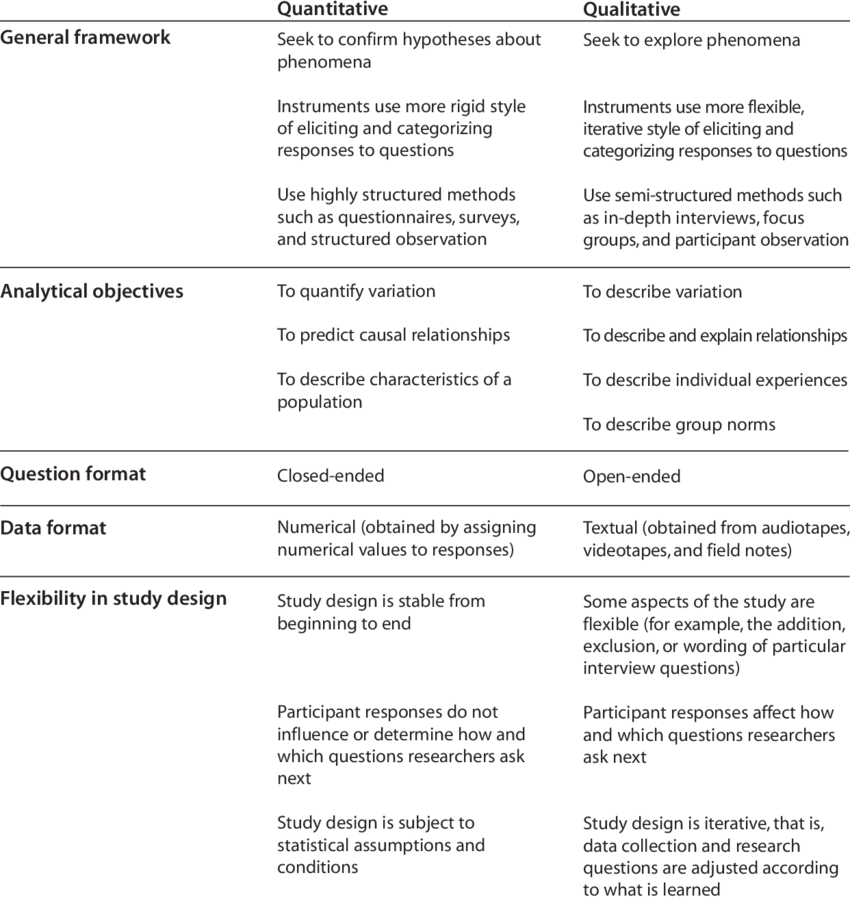
Unlike qualitative research methods, which rely on case studies and surveys and are more subjective, the quantitative approach provides precise, objective results that can be statistically validated as we can see from the above-mentioned analysis. This objectivity is crucial when assessing the effectiveness of AI-driven traffic signal systems, as it enables clear comparisons and predictions based on empirical data. By employing quantitative methods, the study can rigorously evaluate how well the AI algorithms optimize traffic flow, reduce vehicle waiting times, and improve overall traffic management efficiency.

Figure 1 - Quantitative vs Qualitative research methods

## Unique Feature: Object Detection in Traffic Junctions

One of the standout features of our traffic signal control system is the integration of object detection technology at traffic junctions. This section details the methodology behind this unique feature, explaining how it was developed and implemented to detect cars and pedestrians.

### Object Detection

#### Object Detection for Traffic Control

One of the unique features of our traffic light control system is the integration of object detection technology to dynamically detect cars and pedestrians at intersections. This feature leverages the capabilities of real-time image processing to enhance traffic management and safety.

#### Using YOLO for Object Detection

To implement this feature, we use the "You Only Look Once" (YOLO) algorithm, a state-of-the-art object detection framework known for its speed and accuracy. YOLO is particularly well-suited for real-time applications because it processes images in a single pass, making it significantly faster than traditional object detection algorithms that require multiple passes.

#### Theoretical Integration with DQN Model

While it was not possible to implement object detection directly within the simulation environment, a separate task was undertaken to develop the object detection component. This component demonstrates how, in a real-world setup, cameras installed at traffic junctions can use the YOLO algorithm to detect cars and pedestrians.

In theory, the data gathered from these detections would be fed into the Deep Q-Network (DQN) model, which forms the basis of our adaptive traffic light control system. The DQN model would use this real-time information to make informed decisions about traffic light changes, adapting to the current traffic and pedestrian conditions.

#### Conclusion

In theory, object detection at traffic junctions is designed to identify cars and pedestrians in real-time using advanced algorithms, such as YOLO (You Only Look Once). The system processes live video feeds from cameras installed at the junctions, detecting and classifying each object within the frame. This detection data, including the count and position of cars and pedestrians, is then aggregated and passed to the deep Q-network (DQN) model. The DQN model utilizes this information to make informed decisions on controlling traffic lights, optimizing signal timings to improve traffic flow and ensure pedestrian safety. However, due to the limitations of the Simulation of Urban MObility (SUMO) environment, which does not support real-time video feed processing or object detection capabilities, it was not feasible to implement this feature directly into the simulation.

## 3.3 Simulation Environment Development

To effectively evaluate the AI-integrated traffic signal systems, it was crucial to select an appropriate simulation environment. Initially, Unity was considered for its versatility and visual capabilities. However, the decision was made to switch to Simulation of Urban MObility (SUMO) for several compelling reasons, ensuring a more efficient and focused approach to traffic simulation.

#### Advantages of SUMO over Unity

* + - 1. Specialization in Traffic Simulations:

SUMO is specifically designed and optimized for traffic simulations. Unlike Unity, which is a general-purpose game development platform, SUMO provides built-in functionalities tailored for modelling traffic scenarios, including vehicle behaviour, traffic signal control, and pedestrian dynamics. This specialization makes SUMO inherently more suitable for the project's needs.

* + - 1. Scripting and Automation with Python:

SUMO uses Python scripts to run simulations, offering a significant advantage over Unity, which relies on C#. Python's simplicity and readability, combined with extensive libraries and community support, facilitate easier scripting and automation. Given my proficiency in Python and relative inexperience with C#, SUMO presents a more accessible and efficient option for developing and modifying simulation scripts.

* + - 1. Learning Curve and Usability:

The learning curve associated with SUMO is considerably less steep compared to Unity for traffic-specific simulations. SUMO's documentation, tutorials, and user community are focused on traffic management, providing targeted resources that simplify the learning process. This advantage reduces the time and effort required to achieve proficiency in simulation development.

* + - 1. Community and Support:

SUMO has a dedicated user community and extensive support for traffic simulation applications. This includes numerous tutorials, examples, and forums where users can seek advice and share experiences. This robust support network ensures that any challenges encountered during simulation development can be addressed promptly and effectively.

#### Implementation in SUMO

Creating a realistic and functional simulation scenario is a critical step in evaluating AI-based traffic signal control systems. For this project, the Simulation of Urban MObility (SUMO) platform is used to construct and simulate urban traffic networks. The following section outlines the process of creating a network using SUMO's network editor, NETEDIT, and highlights the importance of this step in the overall simulation environment development.

* + - 1. Network Creation in NETEDIT

NETEDIT is a powerful graphical tool within the SUMO suite, designed specifically for creating and editing traffic networks. The process of building a network involves several key steps to ensure that the simulation environment accurately represents real-world traffic conditions.

* + - 1. Setting Up the Basic Network Structure:
* Start NETEDIT: Launch NETEDIT from the SUMO suite.
* Create New Network: Begin by creating a new network file. This serves as the canvas where roads, intersections, and other network elements will be added.
* Add Roads and Intersections: Use the drawing tools to add roads (edges) and intersections (nodes). Each road segment can be defined with specific attributes such as length, number of lanes, speed limits, and lane types (e.g., regular, bus, bike lanes).

#### Experiment Setup: Traffic Intersection Configuration

In this section, we detail the properties of the traffic intersection, including its layout, lane configurations, traffic light setup, and simulation parameters.

* + - 1. Traffic Intersection Configuration

The traffic intersection selected for this experiment is a standard 4-way intersection, featuring arms in the North, South, East, and West directions. Each arm consists of four lanes approaching the intersection and four lanes departing from it.

* Arm Length: The length of each arm is standardized to 500 meters, providing sufficient distance for vehicles to approach and traverse the intersection safely.
* Lane Functionality: On each arm, the right-most lane serves dual functionality, allowing vehicles to either turn right or proceed straight ahead. The two central lanes are designated for vehicles traveling straight ahead, while the left-most lane is exclusively for left-turning vehicles.
* Traffic Light Configuration: Traffic lights are strategically positioned at the intersection to regulate vehicular flow. Each arm's left-most lane has a dedicated traffic light, while the remaining three lanes share a common traffic light. The transition of traffic lights from 'Red' to 'Green' or vice versa is accompanied by a mandatory 'Yellow' phase to ensure smooth transitions and enhance safety.
  + - 1. State-Space

The state-space for this experiment is meticulously designed to capture the dynamic traffic conditions at the intersection. It is constructed based on the configuration of incoming lanes controlled by dedicated traffic lights, divided into lane-groups and lane-cells. The following details the construction of the state-space:

* Lane-Groups: Each incoming lane towards the traffic signal is grouped into lane-groups. These lane-groups facilitate the organization and representation of vehicular flow approaching the intersection.
* Lane-Cells: Within each lane-group, the incoming lanes are further divided into 10 lane-cells. These lane-cells are spatial divisions along the length of the lane, with smaller lane-cells near the intersection and larger ones further away. This differentiation accounts for the varying speeds and densities of vehicles, with closer proximity to the intersection resulting in slower-moving and closely spaced vehicles.
* State-Space Representation: The state-space is represented as a vector of 80 integers, corresponding to the total number of lane-cells across all arms of the traffic intersection. Each element of the vector indicates whether a vehicle is present (1) or absent (0) in the respective lane-cell.

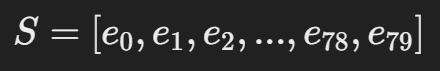
The state-space vector, denoted as 𝑆, is defined as follows:

Figure 2 - State-space vector

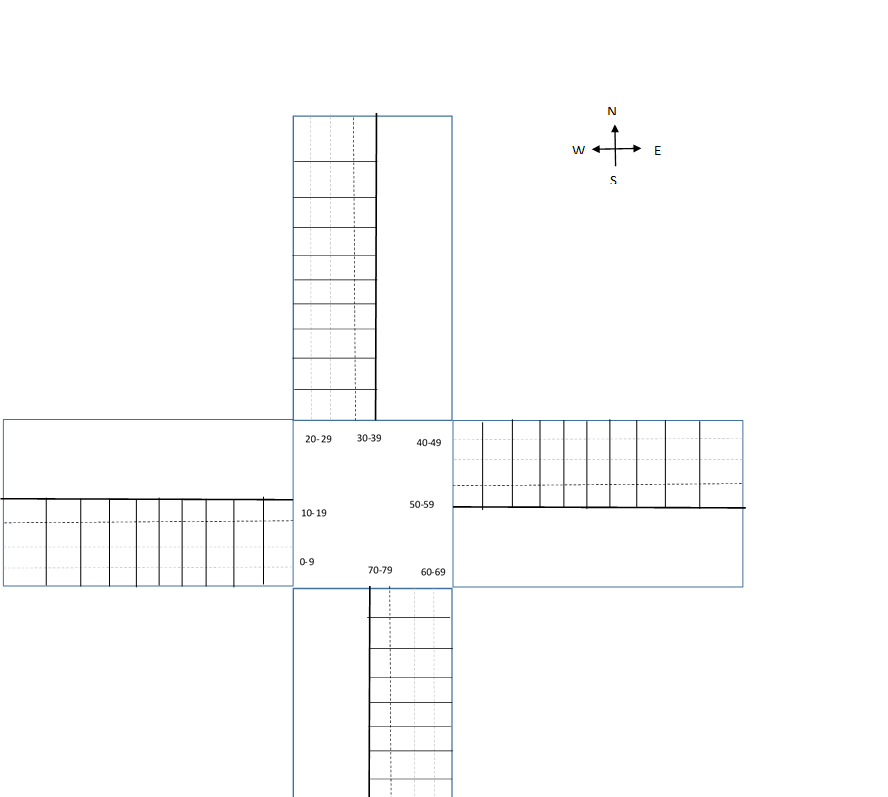
where 𝑒∈{0,1} *e*∈{0,1}, representing the absence (0) or presence (1) of a vehicle in each lane-cell.

Figure 3 - State Space

* + - 1. Action Space

The action space defines the set of all possible actions that the Adaptive TLCS agent can take to control the traffic signals at the intersection. In this experiment, each action corresponds to a specific traffic light phase, representing the permissible states of the traffic signals controlling the intersection. The action space is meticulously defined as follows:

* North South Advance (NSA): This traffic light phase allows traffic to flow from the North direction to the South and from the South to the North. Vehicles in all other directions are halted.

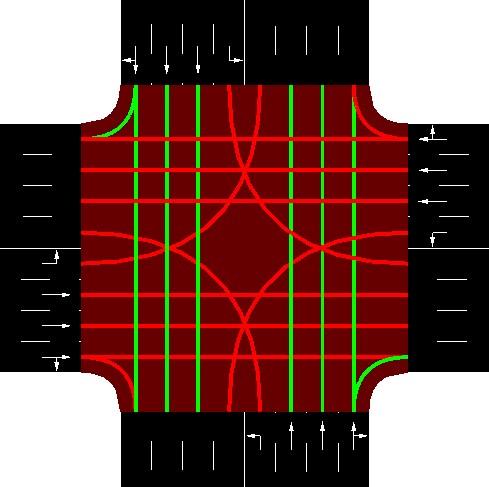


Figure 4 - North South Advance Action Space

* North South Left Advance (NSLA): In this traffic light phase, vehicles are permitted to travel from the North towards the East and from the South towards the West. Traffic flow in all other directions is restricted.

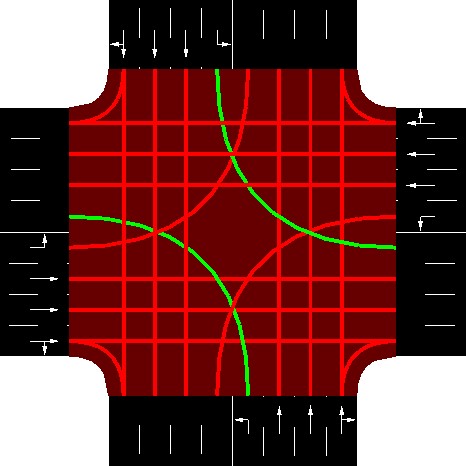


Figure 5 - North South Left Advance Action Space

* East West Advance (EWA): Traffic in this phase can move from the East direction to the West and from the West to the East. All other traffic movements are temporarily halted



Figure 6 - East West Advance Action Space

* East West Left Advance (EWLA): In this traffic light phase, vehicles can proceed from the East to the South and from the West towards the North. Traffic in all other directions is brought to a stop.

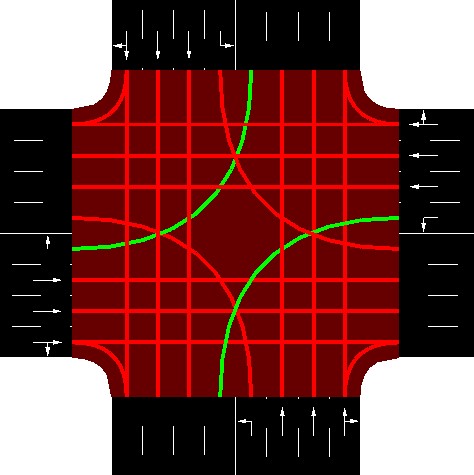


Figure 7 - East West Left Advance Action Space

Each traffic phase is associated with a duration, specified in units of 10 seconds. Additionally, the transition from one traffic phase to another necessitates the preceding phase to undergo a yellow phase for a duration of 4 seconds, ensuring smooth transitions between traffic phases and enhancing safety at the intersection.

By defining the action space with clear and distinct traffic light phases, the Adaptive TLCS agent can effectively control traffic flow at the intersection, optimizing vehicular movement and minimizing congestion.

## DQN Model

The Deep Q-Network (DQN) model serves as the core framework for learning optimal traffic signal control policies in this study. Built upon a Deep Neural Network (DNN) architecture, the DQN model integrates reinforcement learning principles to enable adaptive decision-making in traffic management. The methodology employed for implementing the DQN model is as follows:

* Network Architecture: The DQN model utilizes a DNN architecture consisting of multiple layers of fully connected neural units. The input layer accepts state representations of the traffic environment, while the output layer produces action predictions corresponding to different traffic signal control strategies.
* Feature Extraction: The DQN model extracts relevant features from the input states to capture essential information about traffic flow, vehicle densities, and intersection configurations. These features enable the model to learn meaningful representations of the traffic environment and make informed decisions about traffic signal adjustments.
* Action Selection: At each time step, the DQN model selects actions based on a policy derived from Q-values, which estimate the expected cumulative reward for taking a particular action in a given state. The action with the highest Q-value is chosen as the optimal control strategy for the current traffic conditions.
* Experience Replay: To stabilize training and improve sample efficiency, the DQN model employs experience replay. Experience replay involves storing past experiences (state-action-reward-next state tuples) in a replay buffer and randomly sampling batches of experiences during training. This enables the model to learn from a diverse set of experiences and reduces the impact of temporal correlations in the training data.
* Target Network: To mitigate the issues of overestimation bias and target value instability inherent in Q-learning algorithms, the DQN model incorporates a target network. The target network periodically copies the parameters of the online network, providing stable target Q-values for bootstrapping during training.
* Training Procedure: The DQN model is trained using a variant of the Q-learning algorithm known as Deep Q-Learning. During training, the model iteratively interacts with the traffic environment, observing states, selecting actions, and receiving rewards. The parameters of the DNN are updated iteratively using gradient descent methods to minimize the discrepancy between predicted and target Q-values.

By leveraging the principles of reinforcement learning and deep neural networks, the DQN model autonomously learns effective traffic signal control policies, ultimately enhancing the efficiency and safety of urban traffic management systems.

#### Reward Function

The reward function for evaluating the agent's actions was designed based on the cumulative wait times of all vehicles in the incoming lanes, both before and after an action was taken. If an action facilitated vehicle movement through the intersection, reducing the cumulative wait times, a positive reward was assigned. Conversely, if an action resulted in an increased number of vehicles waiting at the intersection, thereby increasing the cumulative wait times, a negative reward was given.

The reward 𝑅𝑡*Rt*​ at time step 𝑡*t* was calculated using the following formula:

𝑅𝑡=0.9⋅𝑤𝑡−1−𝑤𝑡*Rt*​=0.9⋅*wt*−1​−*wt*​

Here, 𝑤𝑡*wt*​ represents the cumulative wait time (in seconds) of all vehicles in the incoming lanes from the start of the simulation until time step 𝑡*t*. The factor 0.9 is applied to 𝑤𝑡−1*wt*−1​ to stabilize the learning process, ensuring that an action is only considered favourable if it significantly reduces the cumulative wait time compared to the previous time step.

#### Traffic Simulation

To emulate real-life traffic conditions as closely as possible during training, we used a Weibull distribution with a shape parameter of 2. This distribution effectively simulates traffic flow, with vehicle numbers peaking early on (representing rush hour) and then gradually decreasing. Additionally, the simulation was designed such that 75% of the vehicles were set to travel straight, while the remaining 25% were distributed between left and right turns. This realistic traffic generation helps train the model under conditions that closely mimic actual urban traffic patterns.

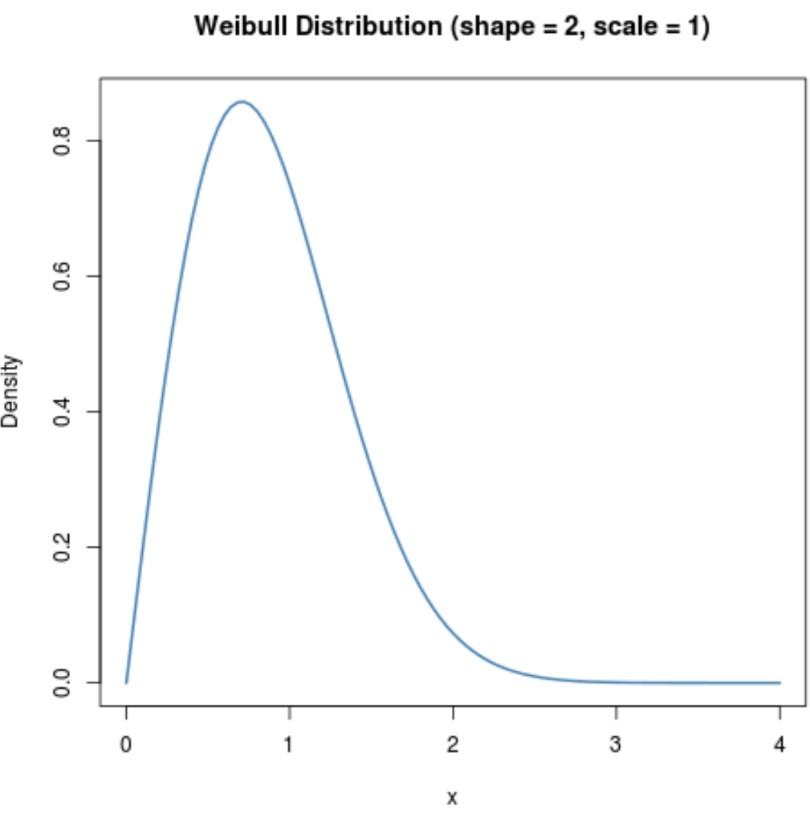


Figure 8 - Weibull Distribution

### Summary

This methodology section ensures that every step of the research process is described in detail, enabling replication. The use of quantitative methods, specific simulation tools, detailed state and action spaces, and a robust reward function collectively contribute to the development and training of an adaptive traffic light control system.

# Chapter 4

# Design and Implementation

This section provides a detailed analysis of our AI-based traffic signal control system's design and implementation. We'll cover the system's architecture, components, and data structure, emphasizing AI integration for traffic flow optimization. Beginning with a conceptual model and system requirements, we'll then delve into the final design, including simulation environment development, state-space/action-space configurations, deep neural network architecture, and reward function for adaptive traffic light control. Each component's role will be thoroughly discussed, offering a comprehensive guide to system design and implementation.

## System Requirements

To run the automated traffic signal simulation system smoothly, certain software and libraries need to be installed and configured. This section outlines the system requirements, including the essential software and libraries utilized in the implementation process. These components are integral to the functioning and performance of the system, ensuring accurate simulation, neural network training, and data visualization.

**Software Requirements:**

* **SUMO (Simulation of Urban MObility):** SUMO is used for traffic simulation and is a core component of the system. It needs to be installed and configured properly.
* **Python:** The implementation is primarily done in Python programming language, requiring Python to be installed on the system. Python 3.x is recommended.

**Libraries and Dependencies:**

* **PyTorch:** PyTorch is utilized for building and training neural network models for reinforcement learning. It's a fundamental library for deep learning tasks in Python.
* **Matplotlib:** Matplotlib is employed for data visualization, enabling the generation of plots to visualize simulation results and performance metrics.
* **NumPy:** NumPy is used for numerical computations and array manipulation, providing essential functionalities for handling data structures and mathematical operations.
* **ConfigParser:** ConfigParser is used for parsing configuration files, facilitating the management of simulation parameters and settings.
* **Sumolib:** Sumolib is a Python library for interacting with SUMO, providing functionalities for configuring SUMO and accessing simulation-related information.
* **Traci:** Traci is another library used for interacting with SUMO, specifically for retrieving simulation data and controlling simulation behaviour programmatically.
* **OS and Sys:** Standard Python libraries like OS and Sys are used for system-related operations, such as file management, path manipulation, and environment variable handling.
* **Timeit:** The Timeit module is utilized for measuring the execution time of specific code segments, aiding in performance analysis and optimization.

## Object Detection using YOLOv8 Algorithm

#### Dataset Preparation:

The BenQ Traffic Dataset was selected for this project due to its diverse range of traffic scenarios and high-quality imagery, which is crucial for training robust object detection models. The dataset, however, required detailed manual annotation to be effectively utilized for training the YOLOv8 object detection algorithm. This annotation process was meticulously conducted using the Computer Vision Annotation Tool (CVAT.ai), an advanced open-source tool designed for annotating images and videos for computer vision tasks.

#### Annotation Process Overview

* + - 1. Initial Setup and Dataset Upload:
* The BenQ Traffic Dataset, comprising thousands of high-resolution images, was first prepared for annotation. Each image captured real-world traffic scenes, including intersections, roads, and various traffic participants such as vehicles and pedestrians.
* These images were uploaded to CVAT.ai, a platform chosen for its robust features and user-friendly interface that streamlines the annotation process. CVAT.ai supports multiple formats and provides various tools to facilitate precise annotation.
  + - 1. Creating Projects and Tasks:

Within CVAT.ai, a new project was created specifically for this dataset. This project was divided into two tasks, training and testing, each containing a subset of images to manage the annotation process efficiently.

* + - 1. Defining Annotation Classes:
* The annotation process began with defining the classes of objects to be annotated. For this project, two primary classes were identified: 'Pedestrian' and 'Car'.
* Each class was associated with a unique colour and label to distinguish them easily during the annotation process. This classification was crucial for training the YOLOv8 model to differentiate between different types of objects accurately.
  + - 1. Drawing Bounding Boxes:
* Annotations had to be manually drawn using bounding boxes around each object of interest within the images. This involved carefully outlining the contours of pedestrians and vehicles to create precise bounding boxes that closely matched the actual shapes and sizes of the objects.
* Pedestrians: For pedestrians, bounding boxes were drawn around individual persons walking, standing, or crossing the road. Particular attention was given to accurately capture the varying postures and movements of pedestrians. Annotators ensured that bounding boxes were neither too tight (excluding parts of the pedestrian) nor too loose (including unnecessary background).
* Cars: For vehicles, bounding boxes were drawn around each car, truck, bus, and motorcycle. The annotation process accounted for different angles and occlusions, such as when vehicles were partially hidden behind others or by environmental features like trees or buildings. Each bounding box was adjusted to tightly fit the visible parts of the vehicle, ensuring consistency and accuracy.
  + - 1. Exporting Annotations:
* Once the annotation process was complete, the annotated data was exported from CVAT.ai in a format compatible with the YOLOv8 algorithm. This typically involved generating text files where each line represented an annotated object, including the class label and the coordinates of the bounding box.
* [image of the training image and labels]
* These files were structured to align with the YOLO format, where each annotation included the image identifier, object class, and bounding box coordinates (x\_center, y\_center, width, height) normalized to the image dimensions.

#### YOLOv8 Algorithm Training

The YOLOv8 algorithm is a cutting-edge model renowned for real-time object detection. Applied here to the BenQ Traffic Dataset, it focuses on identifying pedestrians and cars. This section outlines the training process, covering dataset preparation, training parameter setup, and execution. Overview of YOLOv8 Algorithm: YOLOv8, the latest in the YOLO series, is celebrated for its speed and accuracy. It divides images into grids, predicting bounding boxes and class probabilities directly from features. This direct approach ensures efficient and precise detection, with improvements in architecture and training techniques enhancing performance, ideal for tasks like traffic signal control.

* + - 1. Data Preparation

The annotated BenQ Traffic Dataset was prepared in a format compatible with the YOLOv8 training requirements. This involved organizing the dataset into a directory structure and creating a YAML configuration file to define the training parameters.

* Directory Structure:

The dataset was structured into two main folders: images and labels. The images folder contained the raw images, while the labels folder contained the corresponding annotation files in YOLO format.

* YAML Configuration File:

A YAML file was created to specify the paths to the training and validation datasets, as well as the class names. This file is crucial for the YOLOv8 training process as it informs the model where to find the data and what classes to expect.

The YAML file used for this project was structured as follows:

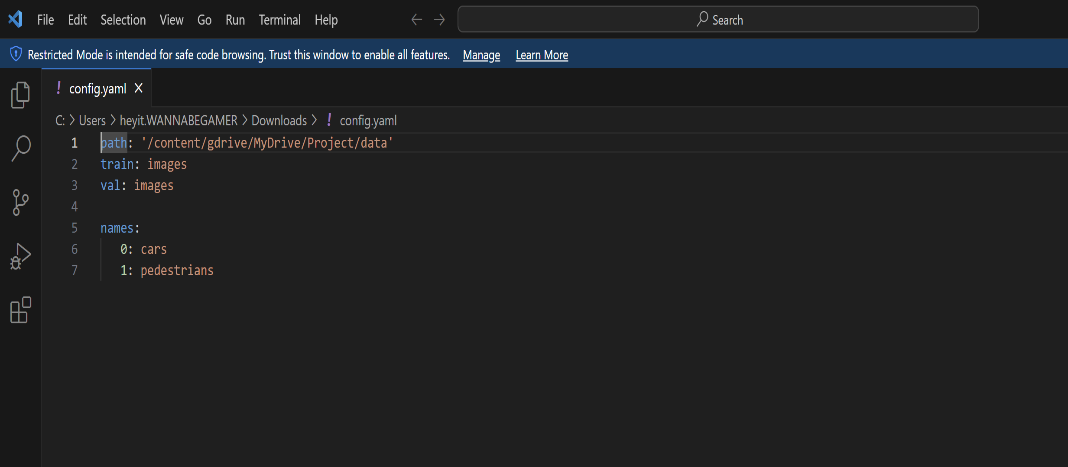


Figure 9 - YAML Configuration File

* + - 1. Training the YOLOv8 Model

The training of the YOLOv8 model was executed using the Ultralytics library, which provides a convenient interface for training YOLO models. The training process involved the following steps:

* Installing the Ultralytics Library

To use the YOLOv8 model, the Ultralytics library needs to be installed. The Ultralytics library provides a convenient interface for working with YOLO models, including model training, evaluation, and inference. To install the Ultralytics library, the following command was used:

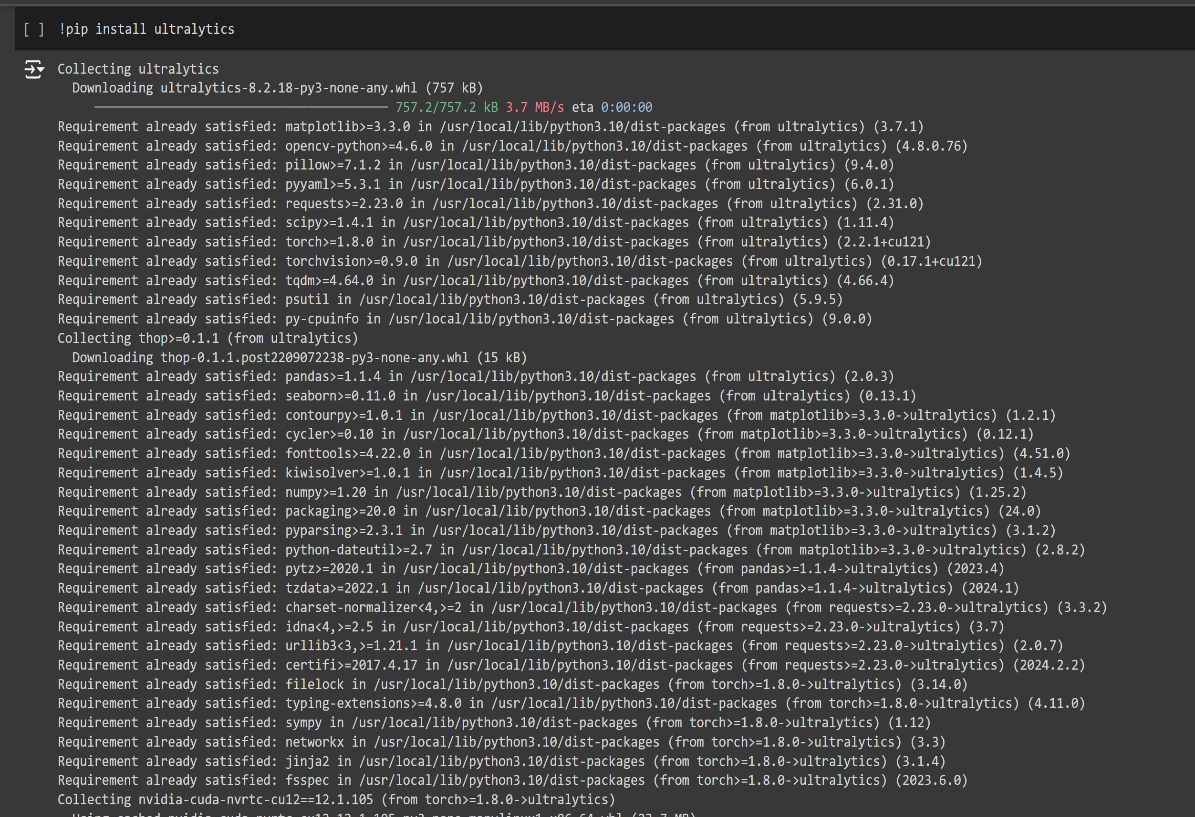


Figure 10 - Installing the Ultralytics Library

* Model Initialization:

The YOLOv8 model was initialized using the yolov8n.yaml configuration file. This file defines the architecture and parameters of the YOLOv8 model.

* Training Configuration:

The training configuration was specified in a Python script, where the YAML file path and the number of epochs were set. The number of epochs was set to 100, ensuring sufficient iterations for the model to learn from the data.

* Execution of Training:

The training process was initiated by executing the script, which involved loading the data, initializing the model, and running the training loop. The script used is shown below:

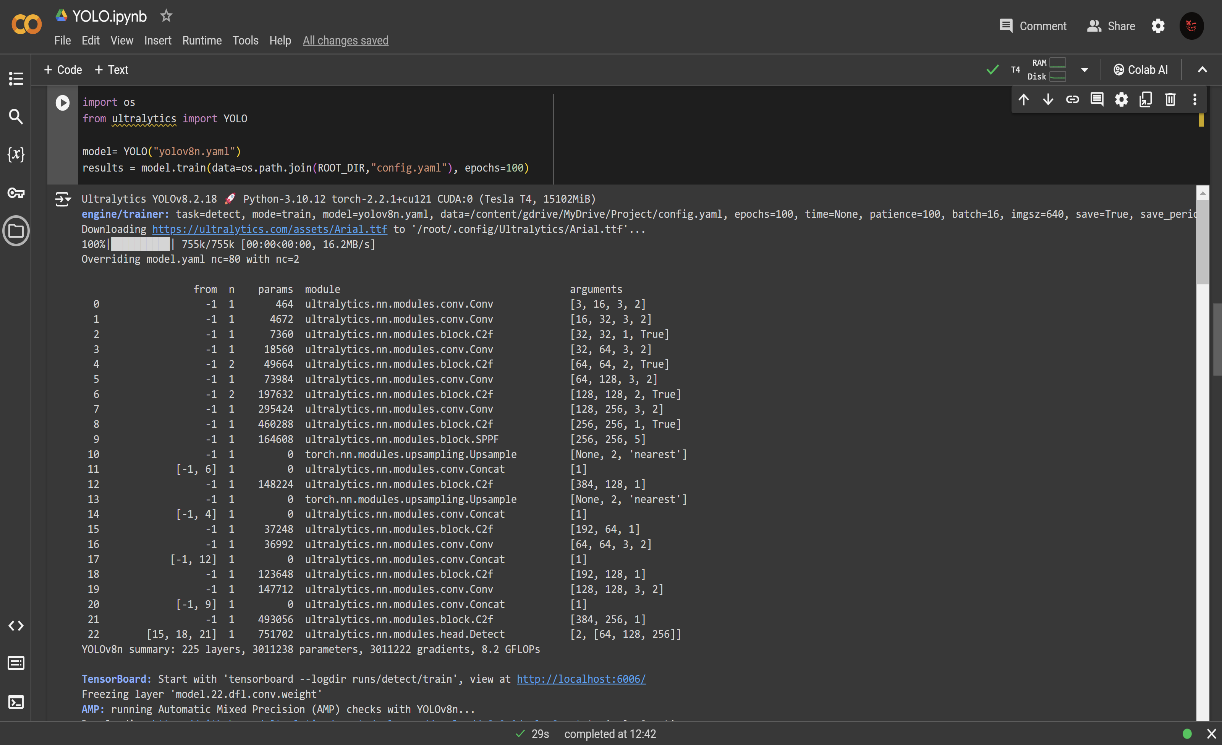


Figure 11 - Initializing the YOLO model and training the dataset

* + - 1. Training Details and Observations

During the training process, the following key details were noted:

* Batch Size and Learning Rate: The model was trained with a batch size and learning rate optimized for the dataset. These hyperparameters were crucial for ensuring that the model learned effectively without overfitting or underfitting.
* Loss Function and Optimization: The YOLOv8 model utilized a combination of classification loss, localization loss, and objectness loss to guide the training process. The Adam optimizer was used to update the model weights based on the gradients computed during backpropagation.
* Training Duration and Performance: Training the model for 100 epochs provided a balance between training time and model performance. Throughout the training process, metrics such as loss, precision, recall, and mean Average Precision (mAP) were monitored to evaluate the model's performance.

## SUMO Simulation Environment Development

The development of the simulation environment is a crucial aspect of this project, as it provides a realistic and controlled setting to test and refine the AI-based traffic signal control system. The Simulation of Urban MObility (SUMO) was chosen as the simulation tool due to its specialization in traffic simulation, extensive support for traffic management studies, and compatibility with Python scripting. This section outlines the process of creating the simulation environment using the SUMO tool, detailing the steps involved in configuring the junction, traffic lights, lanes, and routes.

* + 1. Network Creation with NetEdit

NetEdit, a graphical network editor included with SUMO, was used to design the traffic network. The process involved the following steps:

1. Creating a Simple Junction:
   * A 4-way intersection was created using NetEdit. This intersection represented the traffic junction where the AI-based traffic signal control system would be tested.
   * The intersection was configured with arms extending in the North, South, East, and West directions, each with four lanes approaching and four lanes leaving the intersection.
2. Adding Traffic Lights:
   * Traffic lights were added to the intersection to control the flow of vehicles. Each arm's right-most lane was configured to allow vehicles to turn right or go straight, the two central lanes were set for straight movement, and the left-most lane was dedicated to left turns.
   * A dedicated traffic light was assigned to the left-most lane, while the other three lanes shared a common traffic light. This setup required a mandatory 'Yellow' phase between 'Red' and 'Green' states to ensure safety.
3. Configuring Lanes:
   * The number of lanes was adjusted to match the requirements of the simulation. Each arm of the intersection had a total of four lanes for incoming and outgoing traffic.
   * The lanes were further divided into smaller segments called lane-cells for detailed state-space representation, with lane-cells near the intersection being smaller due to slower and denser vehicle movement.

#### Route Files Configuration

To simulate realistic traffic conditions, route files were created. These files define the paths vehicles take through the network and the timing of their journeys.

1. Generating the Network File (.net.xml):
   * The network created in NetEdit was saved in the .net.xml format, which is compatible with SUMO. This file contains the definitions of the nodes (junctions) and edges (roads) that make up the traffic network.

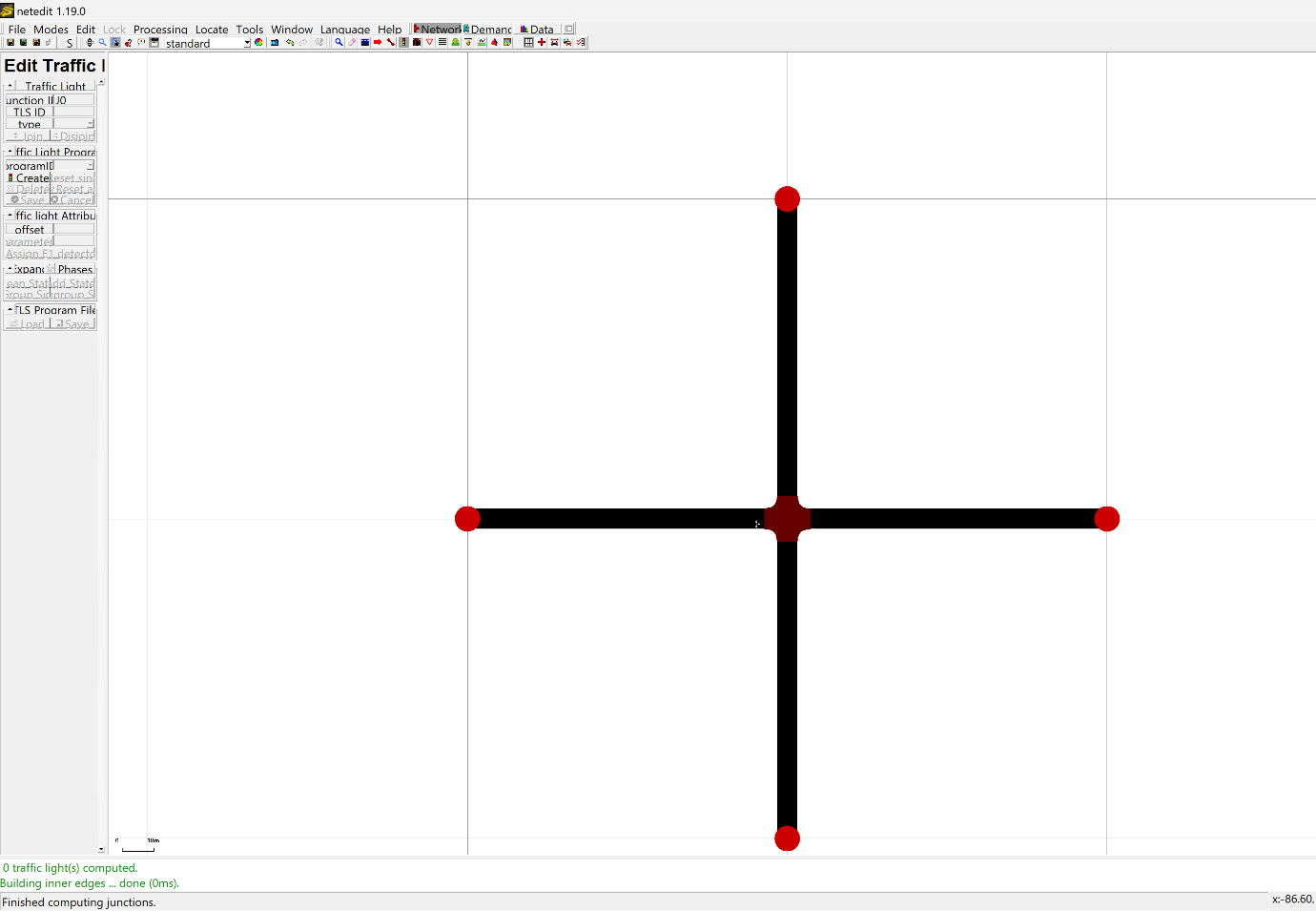
****

Figure 12 - Creating a junction using netedit

1. Creating the Route File:
   * A route file was generated using a Python script. The script defined the departure times, starting points, and destinations for a specified number of vehicles. This file ensures a steady flow of vehicles through the simulation, mimicking real-world traffic patterns.
   * The route file was created based on the trips file, detailing the specific paths each vehicle would take through the network. This file was saved in the ‘. rou.xml’ format and includes the vehicle type, departure time, and route through the network.

#### Implementation Steps

The following steps were taken to implement the simulation environment:

1. Creating the Network:
   * The network was created using NetEdit and saved in the .net.xml format. This file contains the layout of the traffic network, including all junctions, roads, and lanes.

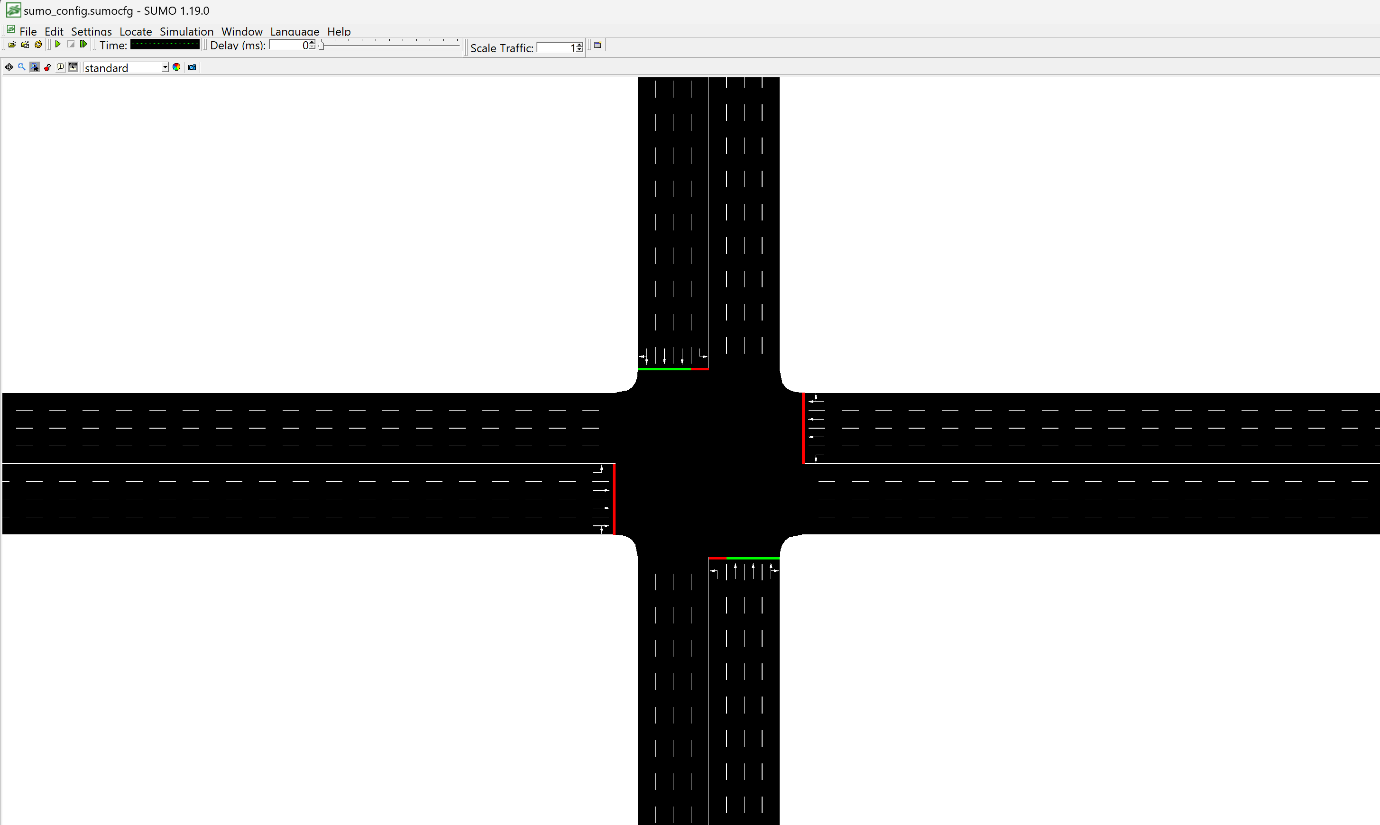
****

Figure 13 - The network created using NetEdit

1. Generating the Route File:
   * A route file was created to define the flow of vehicles through the network. This file includes the starting point, destination, and departure time for each vehicle, ensuring a realistic distribution of traffic over time.
2. Configuration File:
   * A SUMO configuration file (.sumocfg) was created to define the simulation parameters, including the network, trips, and routes files. This file also specified the duration of the simulation and the settings for the traffic lights.

#### Conclusion

The development of the SUMO simulation environment involved meticulous configuration and setup to ensure a realistic representation of urban traffic. By leveraging NetEdit for network design and Python scripts for generating routes files, a robust simulation environment was created. This environment provides a reliable platform for testing and refining the AI-based traffic signal control system, ensuring that the model can effectively manage traffic flow and reduce congestion in real-world scenarios. The choice of SUMO, with its specialized features and strong community support, significantly contributed to the success of this phase of the project.

## Deep Q-Network (DQN) Model for Traffic Light Control

#### DQN Model Overview

A Deep Q-Network (DQN) is a type of reinforcement learning algorithm that combines Q-learning with deep neural networks. It is particularly effective in environments with large state and action spaces. Our DQN model for traffic light control is designed to learn an optimal policy for switching traffic lights by interacting with a simulated traffic environment.

#### Model Architecture

The core of our traffic light control system is a Deep Q-Network (DQN), which is designed to learn an optimal policy for controlling traffic lights by interacting with a simulated traffic environment. The model architecture of our DQN is detailed below:

* + - 1. Neural Network Design

The DQN is implemented as a feedforward neural network using PyTorch. The network architecture is as follows:

* Input Layer:

Input Dimensions: The state representation consists of 32 features (num\_states), capturing various aspects of the traffic conditions at an intersection, such as the number of vehicles waiting at each lane and the duration of the current traffic light phase.

* Hidden Layers:

Number of Layers: The network contains 4 hidden layers (num\_layers).

Layer Width: Each hidden layer has 256 neurons (width\_layers).

Activation Function: Rectified Linear Unit (ReLU) activation functions are used to introduce non-linearity, allowing the network to learn complex patterns in the traffic data.

* Output Layer:

Output Dimensions: The output layer has 4 neurons (num\_actions), corresponding to the Q-values for the four possible actions the traffic light controller can take (e.g., green light for north-south, green light for east-west).

Action Selection: The action with the highest Q-value is selected as the optimal action for the given state.

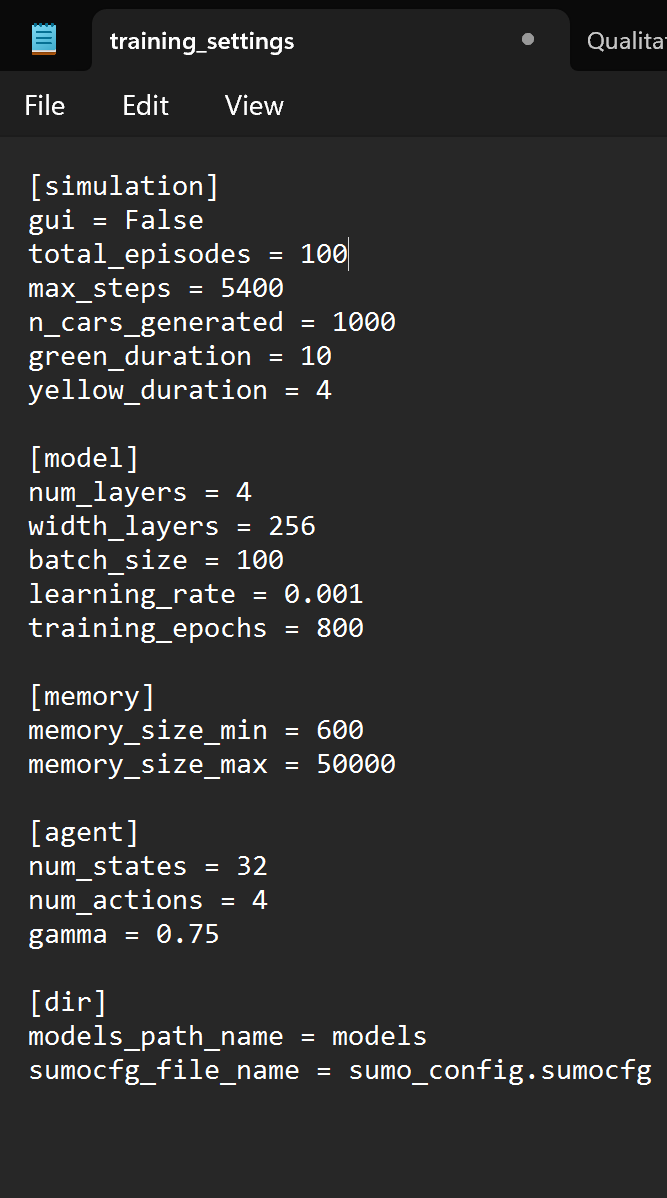
****

Figure 14 - Training Configuration Settings

## Implementation of the AI-Controlled Traffic Signal Simulation

To implement the AI controlled Traffic signal simulation, there are a numerous number of files that need to be executed. Those are:

1. generator.py:

The generator.py script is crucial for generating vehicle routes in the simulation. It creates the episode\_routes.rou.xml file, dictating vehicle paths for each simulation episode. Using numpy and math libraries, it mimics real-world traffic patterns based on a Weibull distribution, ensuring realistic vehicle arrivals. This variability adds realism to traffic flow, vital for training the AI model. The script assigns vehicles diverse routes, ensuring comprehensive simulation of traffic movements. Additionally, the SUMO network (.net.xml) and route files (.rou.xml) are essential. The network file defines road layout, while the route file specifies vehicle paths. Together with the SUMO configuration file, they ensure accurate simulation. The DQN model code interfaces with SUMO, enabling AI-driven traffic optimization. This setup forms a robust platform for developing and testing advanced traffic management solutions.

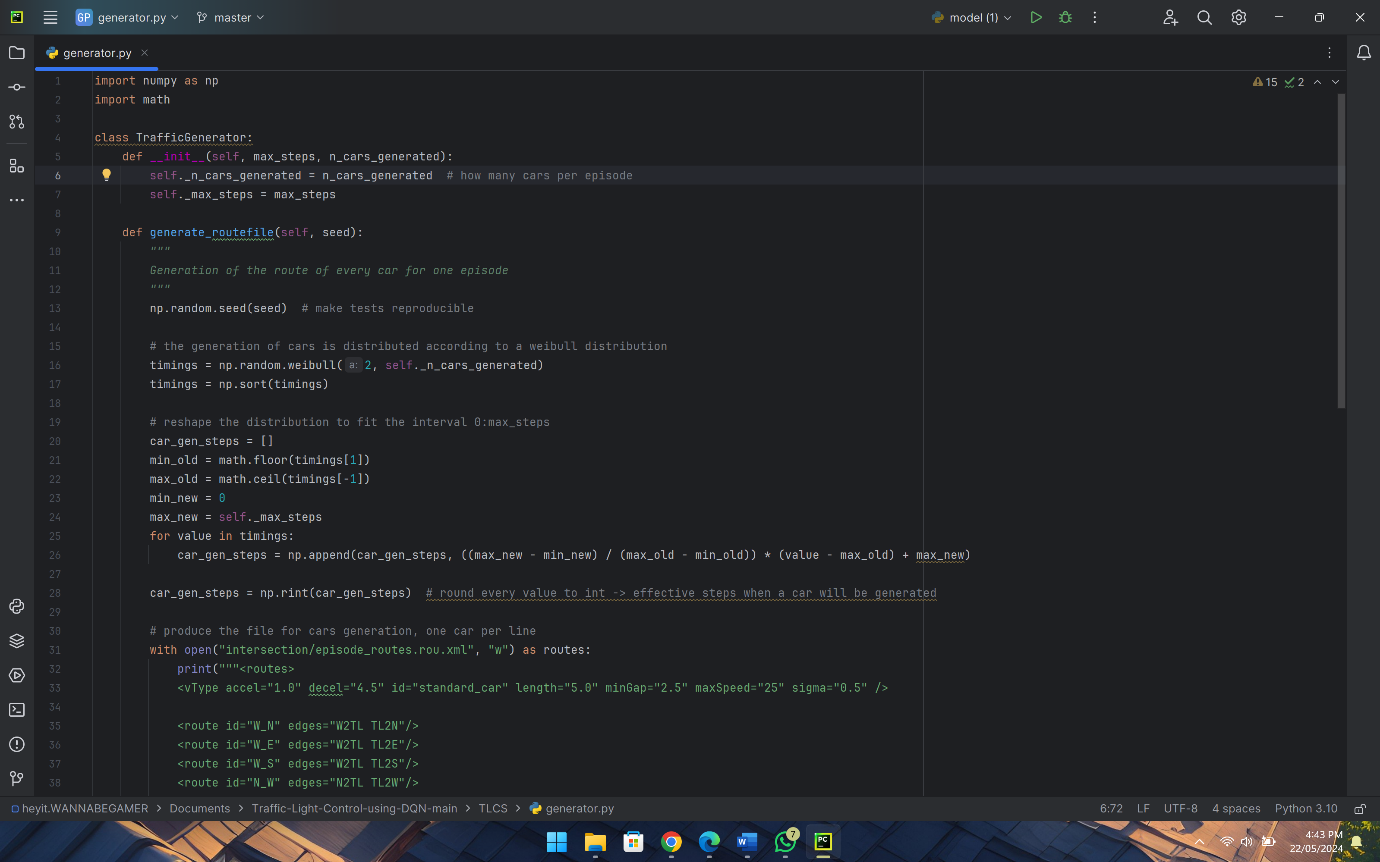


Figure 15 - generator.py file

1. memory.py

The memory.py file is integral to the reinforcement learning framework of the AI-controlled traffic signal simulation. It manages past experiences efficiently, balancing resource usage. The add\_sample method seamlessly integrates new experiences while retaining relevant ones. The get\_samples method retrieves samples for training, ensuring diversity. Overall, memory.py enhances the AI agent's learning and decision-making, optimizing traffic flow in the simulation

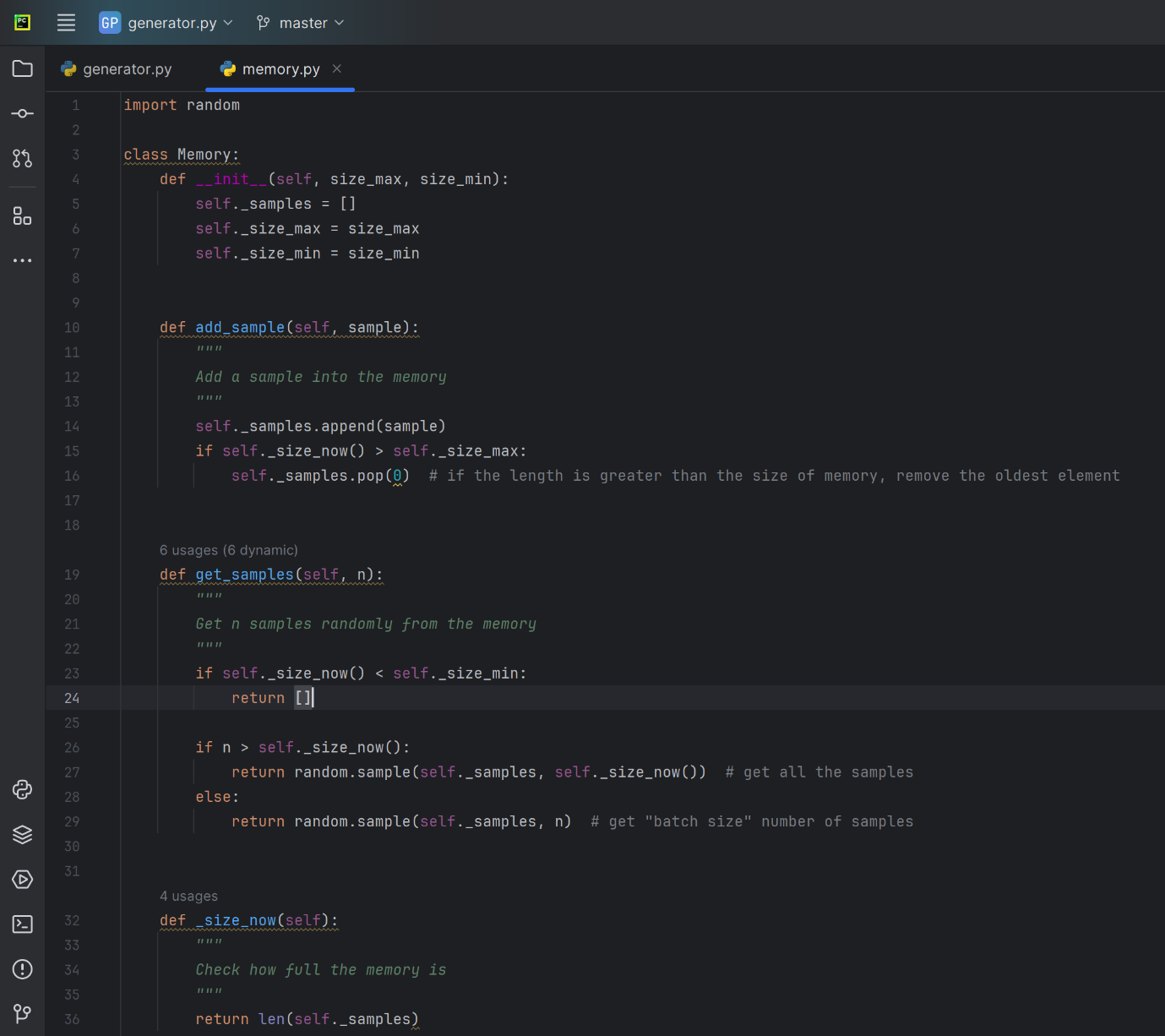


Figure 16 - memory.py file

1. model.py

The model.py file defines the neural network architecture for training the AI agent in the traffic signal simulation. The TrainModel class initializes parameters like layers, batch size, and learning rate, allowing for customizable configurations. The model's layers include input, hidden, and output layers, each undergoing linear transformations with ReLU activation for non-linearity. Training utilizes the Adam optimizer and mean squared error loss function. Leveraging hardware capabilities, the model efficiently handles training and inference tasks. Through the forward method, input states are processed for action predictions. In essence, model.py is pivotal in enabling the AI agent's learning and decision-making in the traffic simulation.

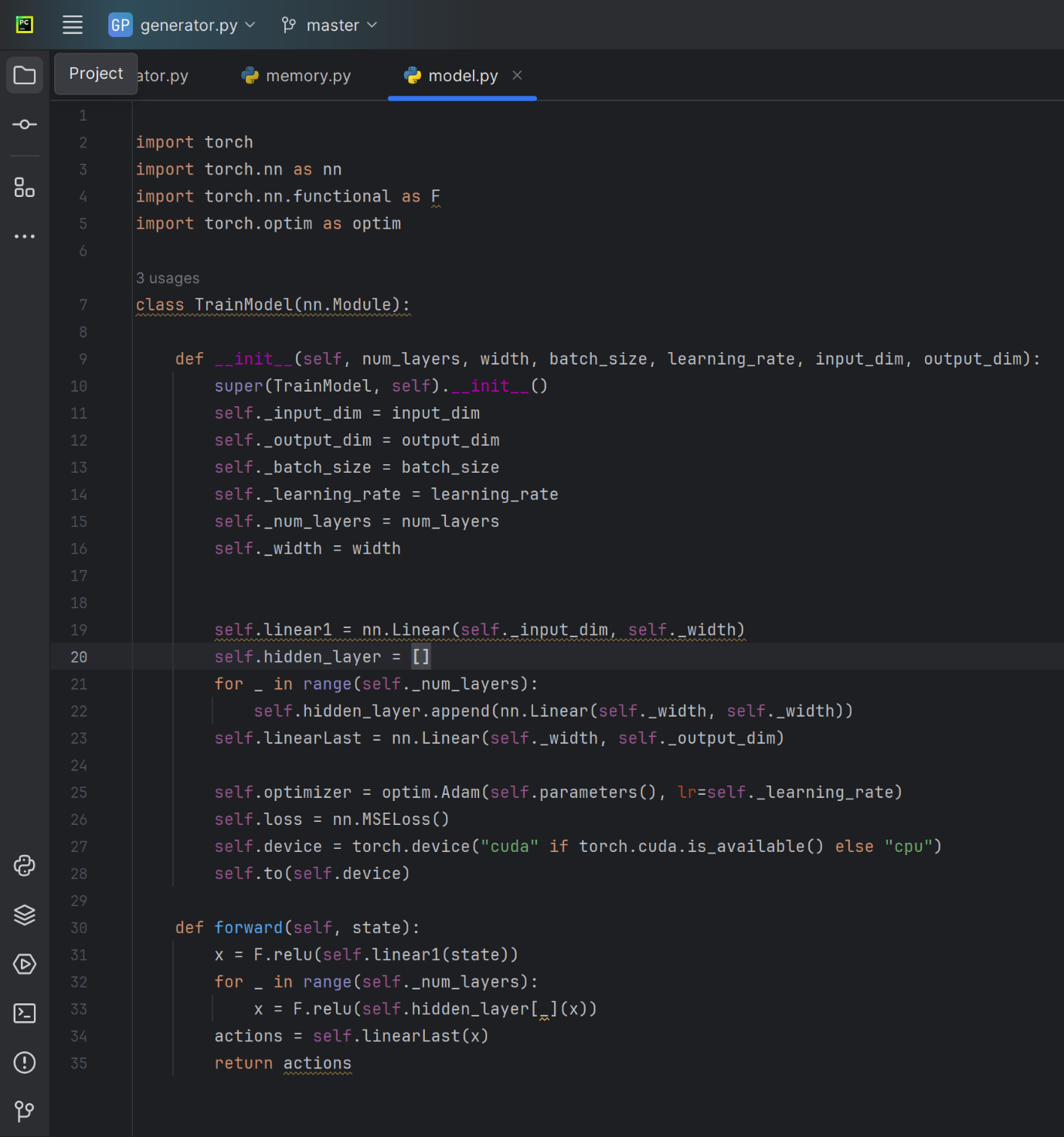


Figure 17 - model.py file

1. testing\_main.py

The testing\_main.py file manages traffic signal simulation and evaluation using a trained AI model. It loads settings from testing\_settings.ini, sets up paths, and initializes classes for model testing, traffic generation, and visualization. The Simulation class integrates the model, traffic generator, and SUMO configuration for simulation. Results are printed after executing simulation episodes, and relevant information and plots are saved. Overall, testing\_main.py orchestrates the testing process for the AI-controlled traffic signal system.

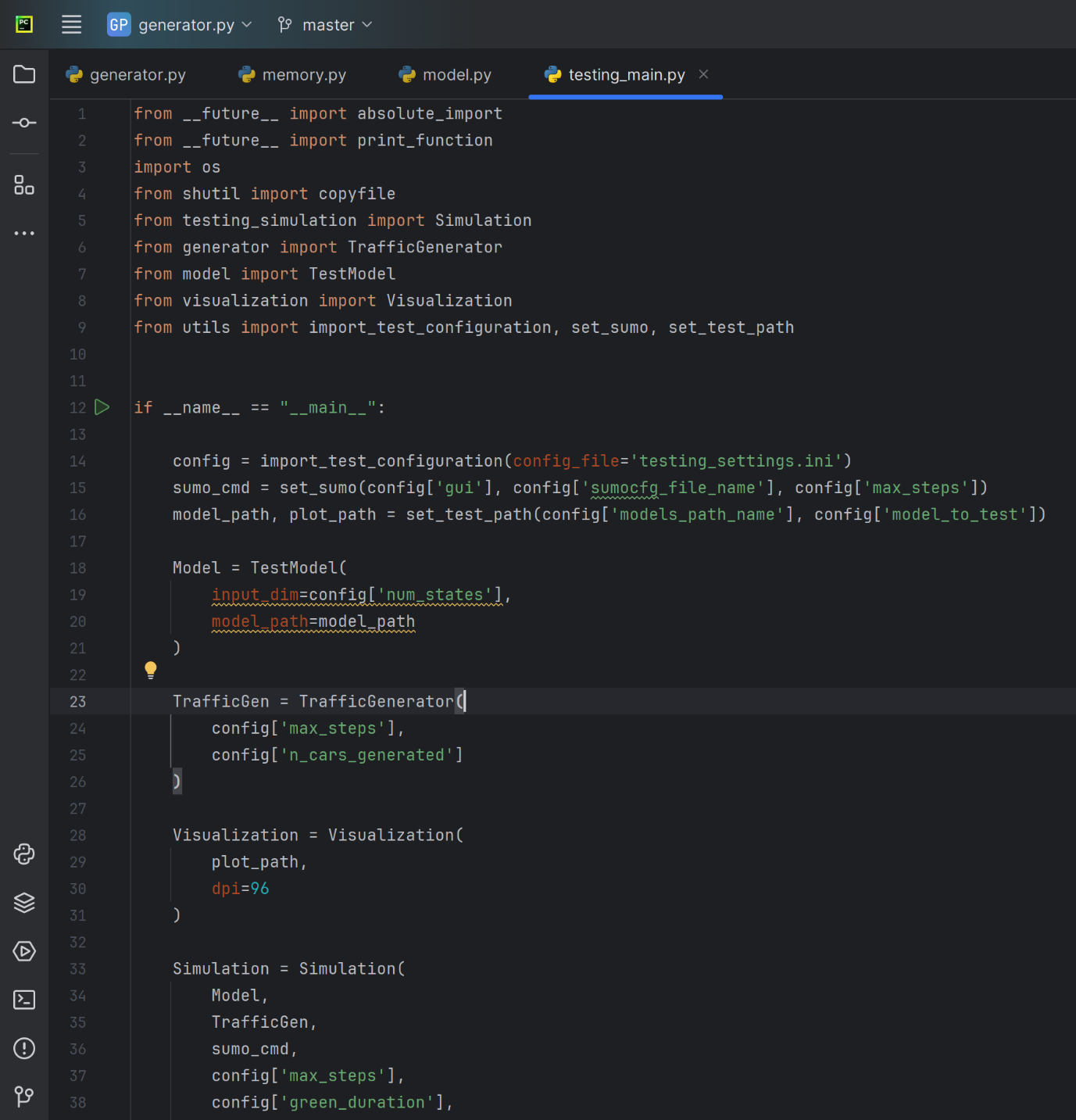


Figure 18 - testing\_main.py file

1. testing\_settings.ini

The testing\_settings.ini configuration file serves as a crucial blueprint for orchestrating the evaluation and testing of the AI-managed traffic signal system. It encapsulates vital simulation parameters such as GUI display preference, maximum simulation steps per episode, and the number of generated cars. Additionally, it specifies agent-related details like the number of states and available actions for traffic signal control. Furthermore, directory paths for storing trained models and SUMO configuration files are defined. By configuring these parameters, the file facilitates the reproducible execution of simulation episodes and the evaluation of AI agent performance, playing a pivotal role in optimizing and refining the traffic management system.

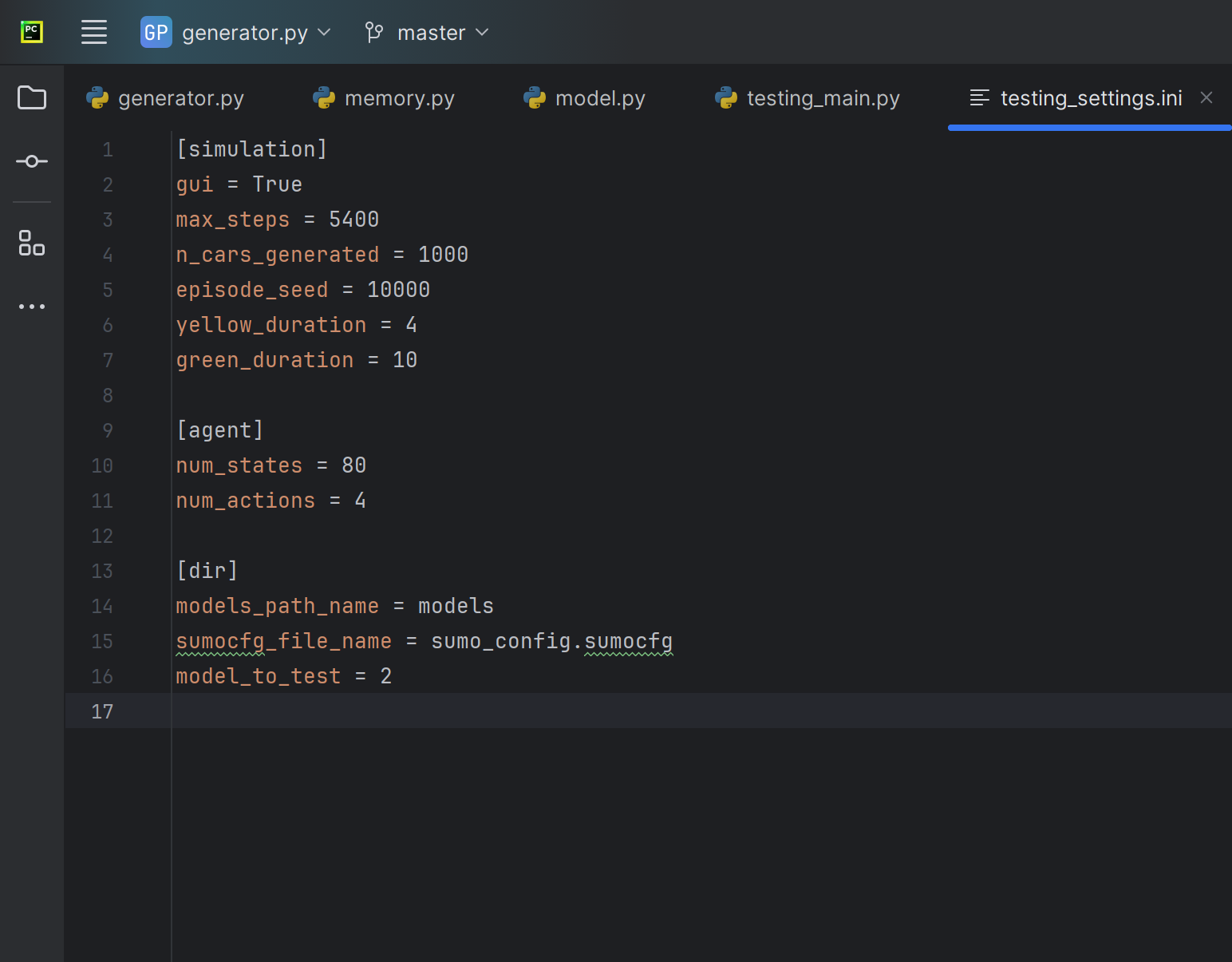


Figure 19 - testing\_settings.ini file

1. testing\_simulation.py

The testing\_simulation.py file evaluates the performance of an AI-controlled traffic signal system using SUMO simulation software via the TraCI API. Orchestrated by the Simulation class, it manages traffic signal phases, computes rewards, and interacts with the SUMO environment. Parameters like model, traffic generation, and simulation settings are passed to initialize the Simulation class. During simulation, the run method coordinates model-SUMO interaction, selecting signal phases based on predictions, and collecting data on queue lengths and rewards. The class utilizes methods to interact with SUMO, transition phases, set light states, and assess congestion. Predefined constants represent signal phases, and helper methods convert lane positions. Properties access queue length and reward data for analysis. Overall, this script is vital for testing AI-driven traffic signal control strategies in a simulated urban environment

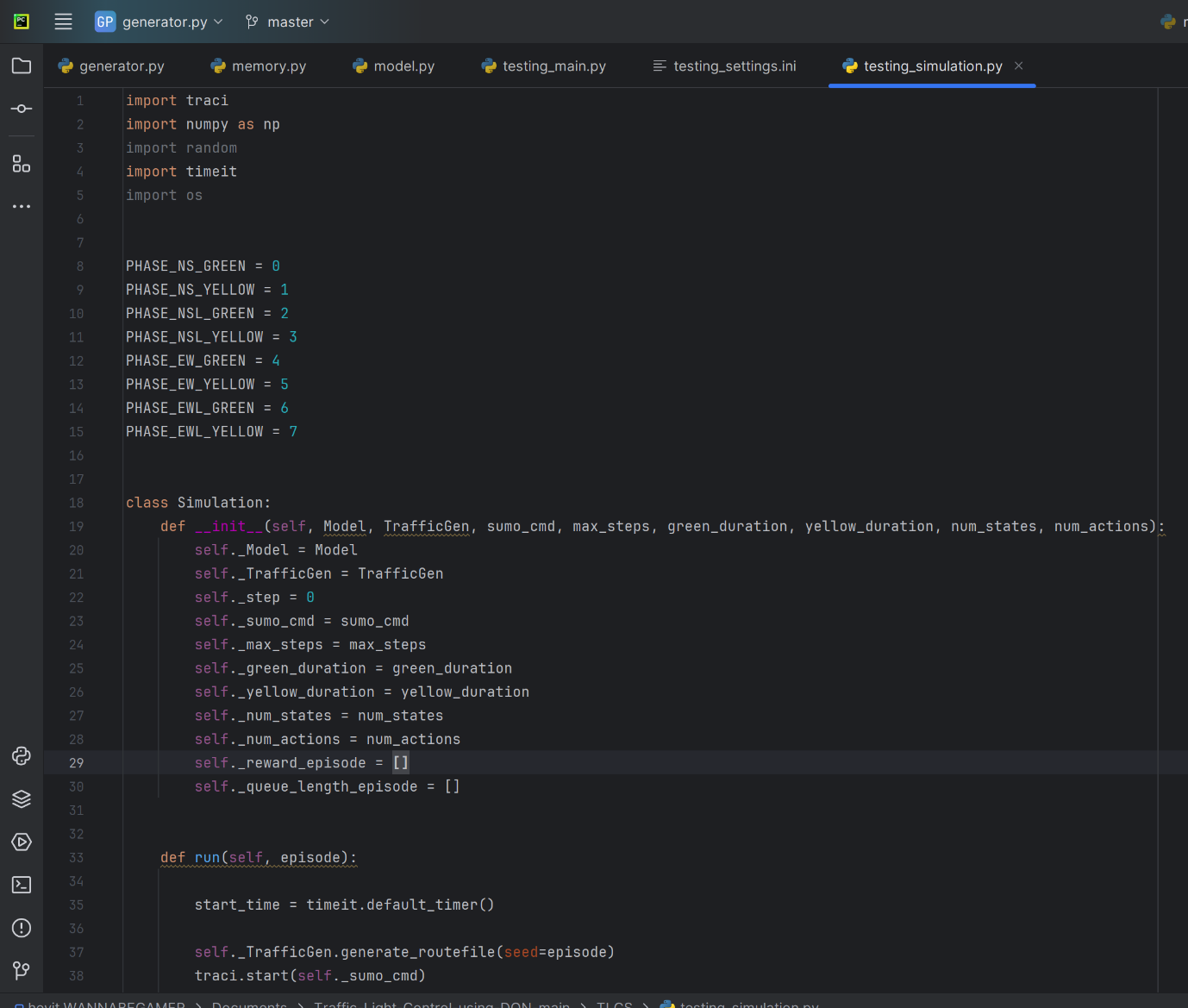


Figure 20 - testing\_simulation.py file

1. training\_main.py

The training\_main.py file is the entry point for training the AI-controlled traffic signal system, orchestrating the entire process. It imports configurations from training\_settings.ini, initializes the SUMO simulation environment, sets up paths for file saving, and instantiates components like the neural network model, memory buffer, traffic generator, and visualization module.

The training loop runs episodes, where each involves simulation with epsilon-greedy exploration, data collection, and model training. The script records simulation and training times for each episode. After completing all episodes, it saves the trained model, configuration file, and session information. Plots of key metrics are also saved for visualization and analysis.

Executing training\_main.py initiates training, resulting in a trained model capable of managing traffic signals effectively in a simulated urban environment.

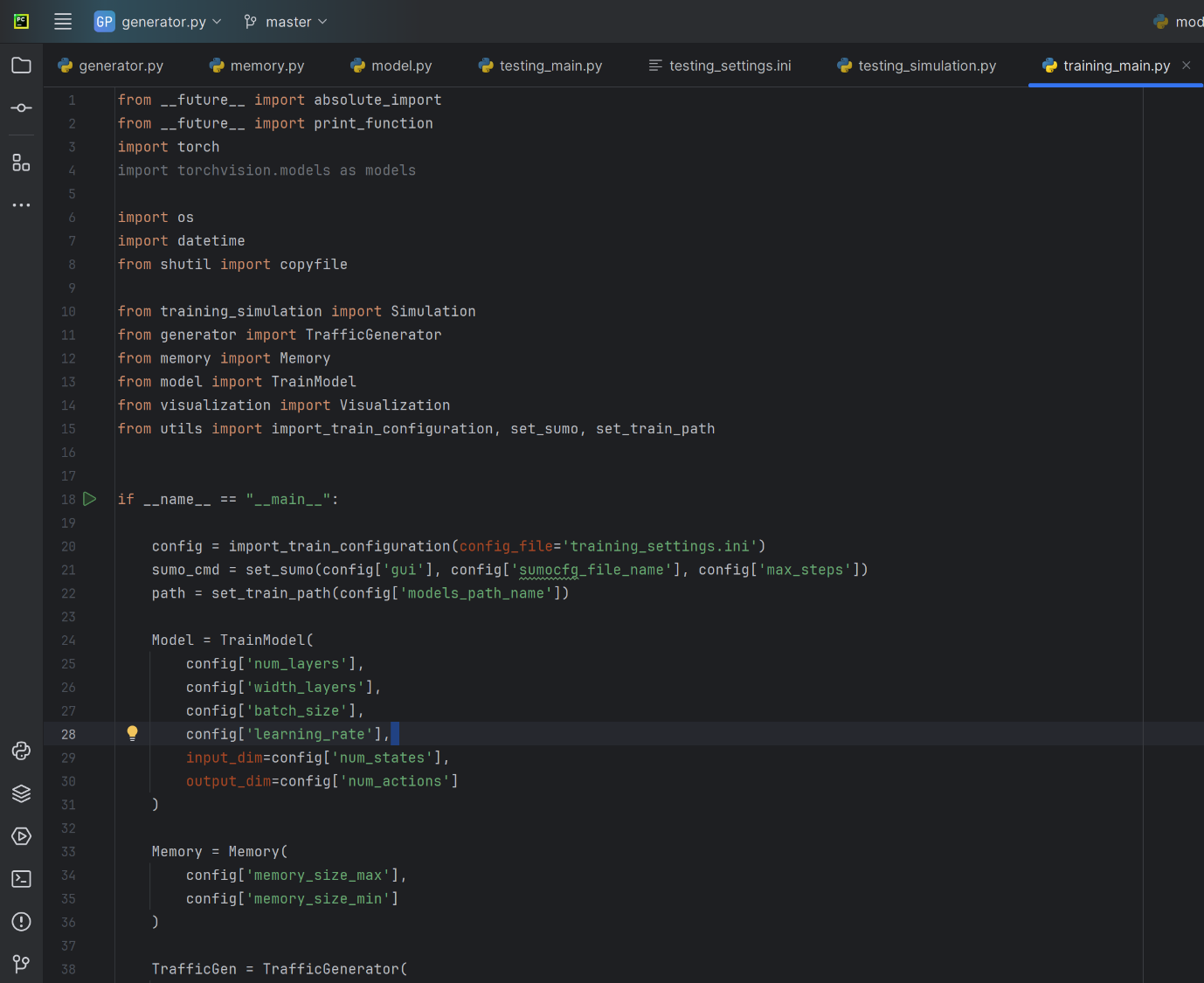


Figure 21 - training\_main.py

1. training\_settings.ini

The training\_settings.ini file serves as a configuration file for the AI-controlled traffic signal system's training process. It includes parameters categorized into sections like simulation, model, memory, agent, and directory settings. These parameters cover simulation environment settings, neural network model architecture, batch size, learning rate, training epochs, memory sizes, agent parameters, and directory settings for model and SUMO configuration files. Overall, this configuration file governs the training process and behaviour of the AI-controlled traffic signal system.



Figure 22 - training\_settings.ini file

1. training\_simulation.py

The training\_simulation.py file implements the simulation environment for training the AI-controlled traffic signal system. It defines a Simulation class orchestrating interactions between SUMO traffic simulator, neural network model, and memory buffer for experience replay during training. The code initializes parameters and settings, executes simulation and training loops for episodes, interacts with SUMO for traffic state information, selects actions based on an epsilon-greedy policy, updates memory buffer, and trains the neural network using experience samples. It records statistics like cumulative negative reward, waiting time, and queue length for each episode, facilitating monitoring of training progress. This code encapsulates logic for training the AI agent to learn effective traffic signal control strategies through simulation and reinforcement learning.



Figure 23 - training\_simulation.py

1. utils.py

The utils.py file serves as a collection of utility functions designed to streamline various aspects of automated traffic signal simulation and management of training/testing processes

for AI-controlled traffic signal systems. These utilities encompass functionalities such as importing configuration settings for training and testing, configuring SUMO parameters for simulation, organizing paths for storing model checkpoints and test results, and ensuring efficient management of model versions. By encapsulating these common tasks into reusable functions, utils.py enhances the automation and efficiency of traffic signal simulation workflows, enabling smoother development and evaluation of AI-based traffic control algorithms.

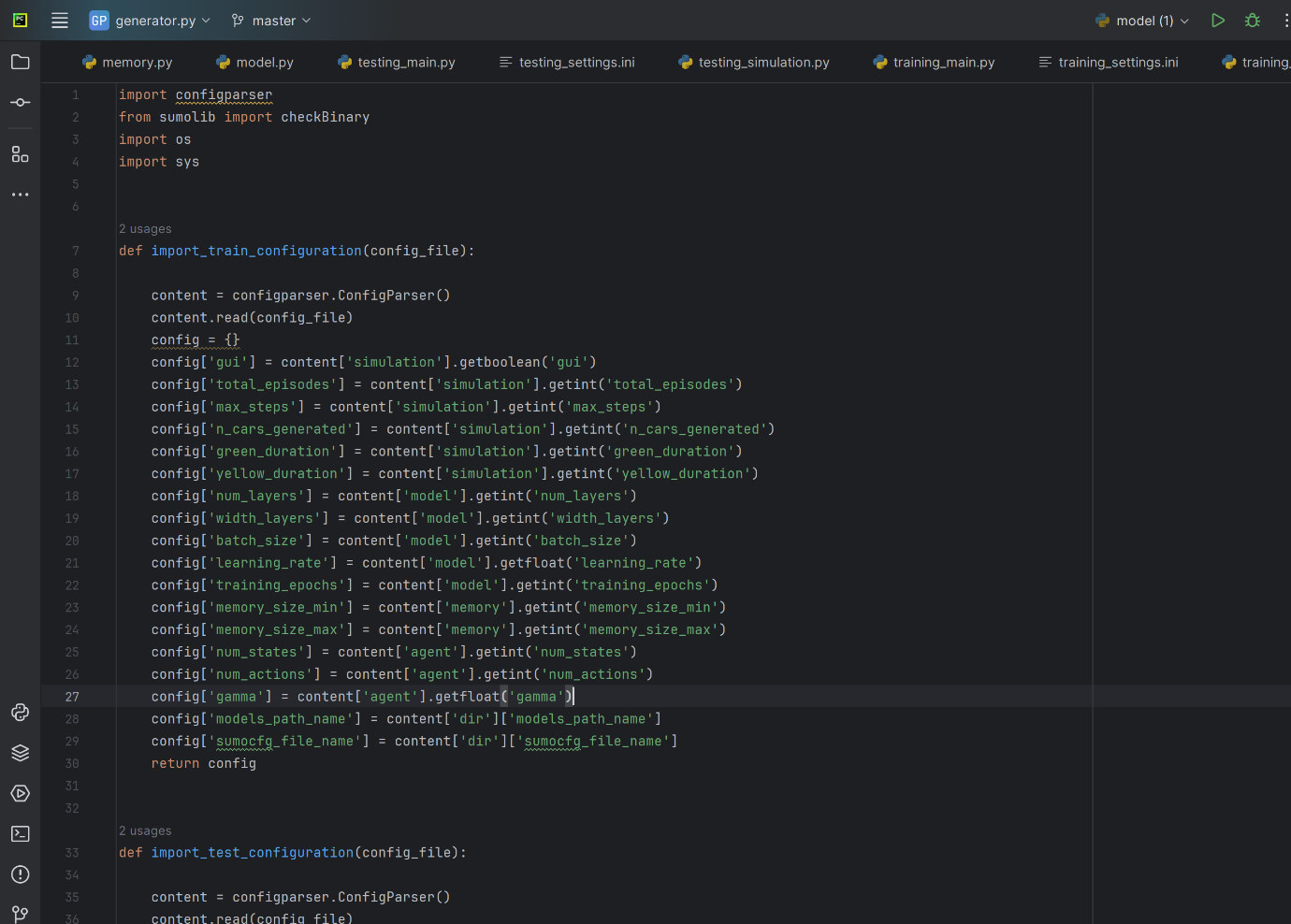


Figure 24 - utils.py file

1. visualization.py

The Visualization class in visualization.py facilitates the generation and storage of visualizations for traffic signal simulation data. Its save\_data\_and\_plot method generates plots using matplotlib, saving them as PNG images along with the data used for the plot in a text file. This functionality allows users to analyse key metrics like cumulative negative reward, delay, and queue length, supporting the refinement of AI-based traffic signal control strategies.

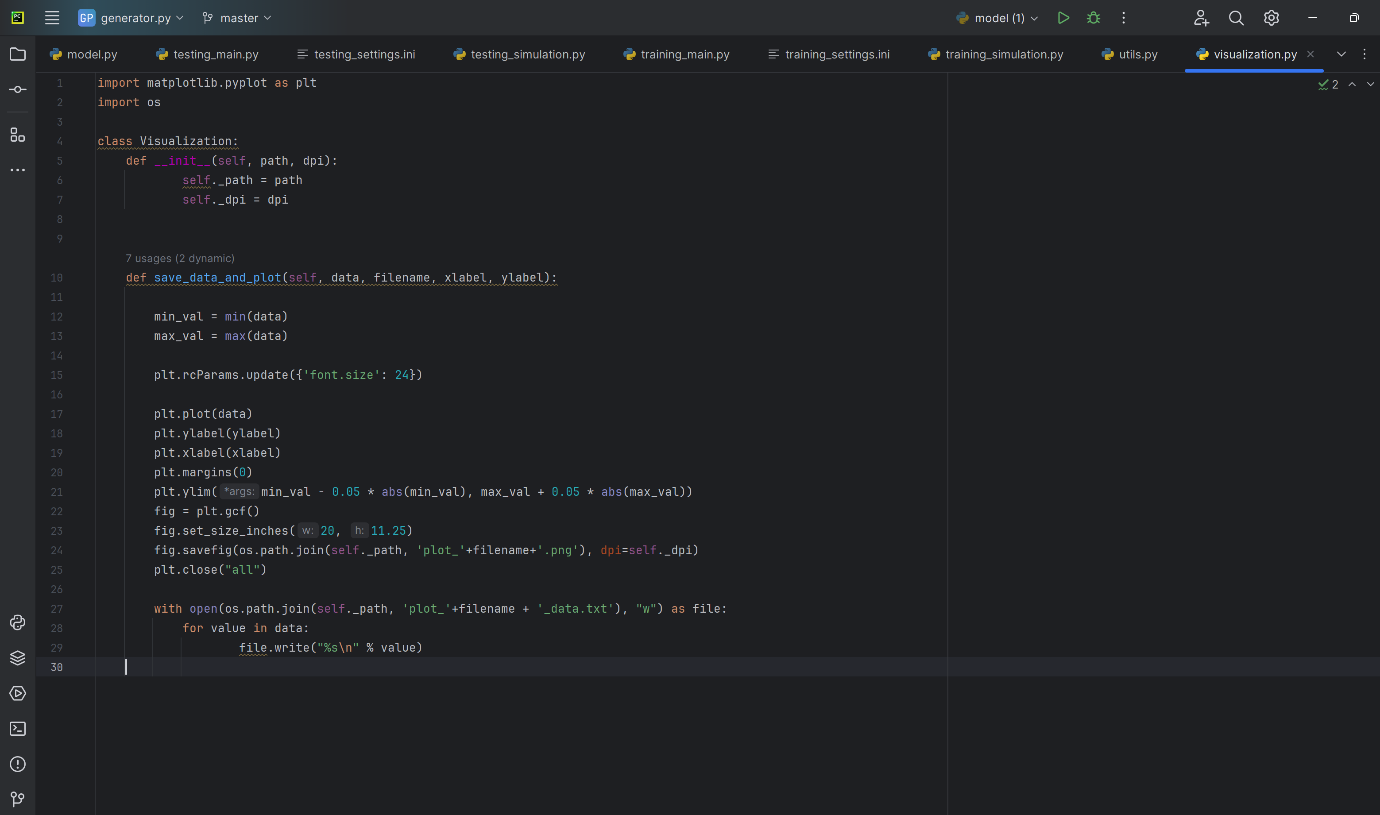


Figure 25 - visualization.py file

## Conclusion

In conclusion, the traffic signal simulation system presented in this report offers a comprehensive solution for studying and optimizing traffic flow management using reinforcement learning techniques. Comprising interconnected components like training\_main.py, training\_simulation.py, utils.py, and visualization.py, each fulfils specific roles.

training\_main.py orchestrates training, leveraging training\_simulation.py for simulation and data collection. utils.py configures parameters and handles simulation setup, while visualization.py generates performance visualizations.

This modular design ensures flexibility, scalability, and ease of maintenance, empowering experimentation with traffic control strategies and adaptation to various scenarios. Integrating reinforcement learning, simulation, and visualization, this system is a valuable tool for advancing traffic management research and developing intelligent transportation systems to enhance traffic efficiency and safety.

# Chapter 5

# Result and Analysis

## 5.1 Object Detection Results

The object detection task was implemented using the YOLO (You Only Look Once) algorithm to detect cars and pedestrians. This section presents the results of the YOLO model's training process, highlighting the key metrics and their implications for the model's performance. In this context, the number 0 denotes cars and 1 denotes pedestrians.



Figure 26 - Results of YOLO object detection



Figure 27 - Results of object detection using YOLOv8

the graphs below illustrate the model's training and validation performance over 100 epochs:

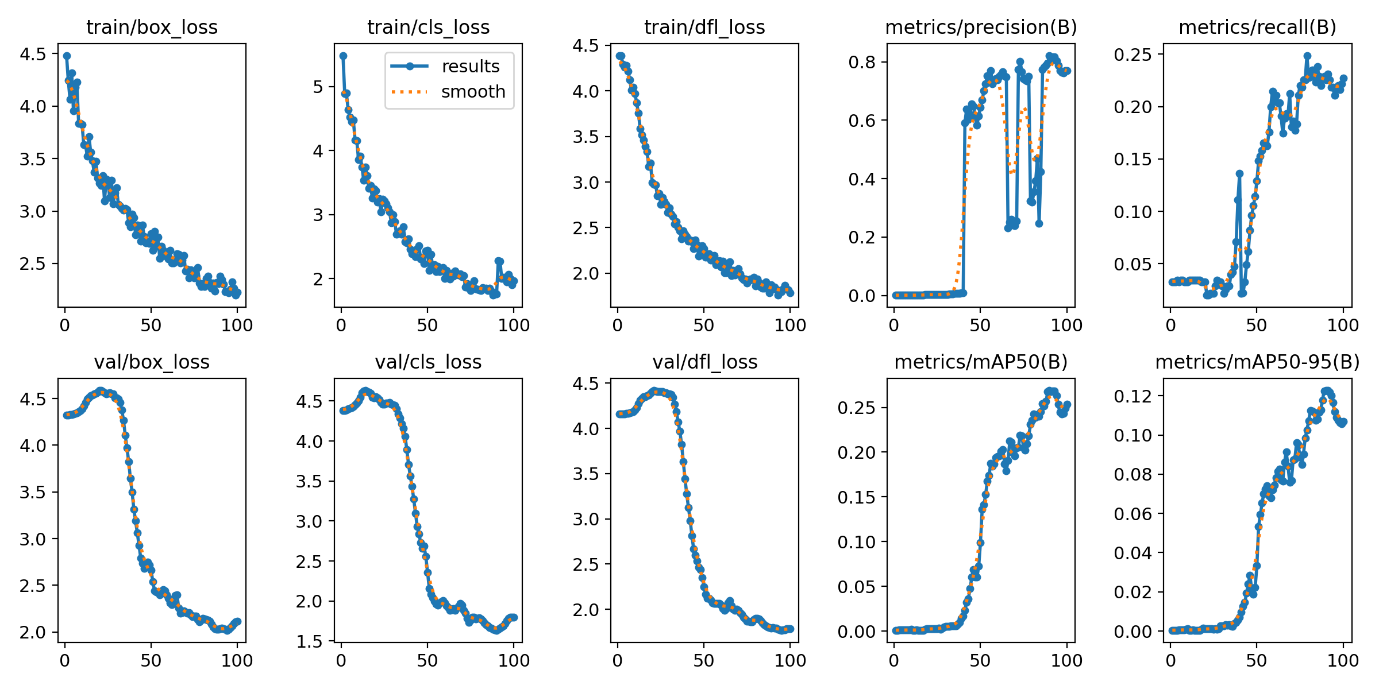


Figure 28 - Training results of the dataset using YOLOv8

The training results of the YOLO object detection model, as depicted in the provided graphs, indicate several key inferences:

Training Losses:

train/box\_loss: This graph shows a consistent decrease in the bounding box regression loss during training, indicating that the model is learning to predict bounding box coordinates more accurately over time.

train/cls\_loss: The classification loss also decreases steadily, suggesting that the model is improving in correctly classifying the detected objects as either cars (0) or pedestrians (1).

train/dfl\_loss: The distributional focal loss (DFL) is reducing, which signifies that the model's overall object detection performance is improving during training.

Validation Losses:

val/box\_loss: The validation box loss decreases, although there is a noticeable initial fluctuation before it stabilizes. This indicates that the model is generalizing well to unseen data regarding bounding box predictions.

val/cls\_loss: Similar to the training classification loss, the validation classification loss shows a consistent downward trend, implying good generalization in object classification.

val/dfl\_loss: The validation DFL decreases steadily, mirroring the training DFL, which suggests robust model performance on unseen validation data.

Precision and Recall:

metrics/precision(B): Precision fluctuates initially but eventually increases, indicating that the proportion of true positive detections out of all positive detections is improving.

metrics/recall(B): Recall shows a clear upward trend, reflecting an increase in the proportion of true positive detections out of all actual positives in the validation set.

Mean Average Precision (mAP):

metrics/mAP50(B) and metrics/mAP50-95(B): Both mAP metrics show an increase over the training epochs, demonstrating that the overall object detection accuracy, considering both precision and recall, is improving. mAP50 measures the model's performance at an IoU threshold of 50%, while mAP50-95 averages the performance over multiple IoU thresholds.

## 5.2 Training Results for the DQN Model

The training results of the Deep Q-Network (DQN) model offer insights into its performance and learning capabilities, crucial for evaluating traffic signal control optimization. Through reinforcement learning, the model undergoes extensive training with fine-tuned parameters for enhanced convergence and efficiency. Metrics like cumulative negative reward, delay, and queue length across episodes provide performance evaluation, complemented by visualizations like reward plots and queue length trends, offering a comprehensive view of training dynamics and convergence.

5.2.1 Cumulative Delay

Cumulative Delay: Cumulative delay refers to the total delay experienced by vehicles in the traffic network over a specific period, typically measured in seconds or vehicle-hours. It accounts for the additional time vehicles spend traveling due to congestion, traffic signals, or other factors (Wang et al., 2020). Cumulative delay is an important metric for assessing the overall efficiency and performance of a traffic control system. Lower cumulative delay indicates better traffic flow and reduced travel time for vehicles.

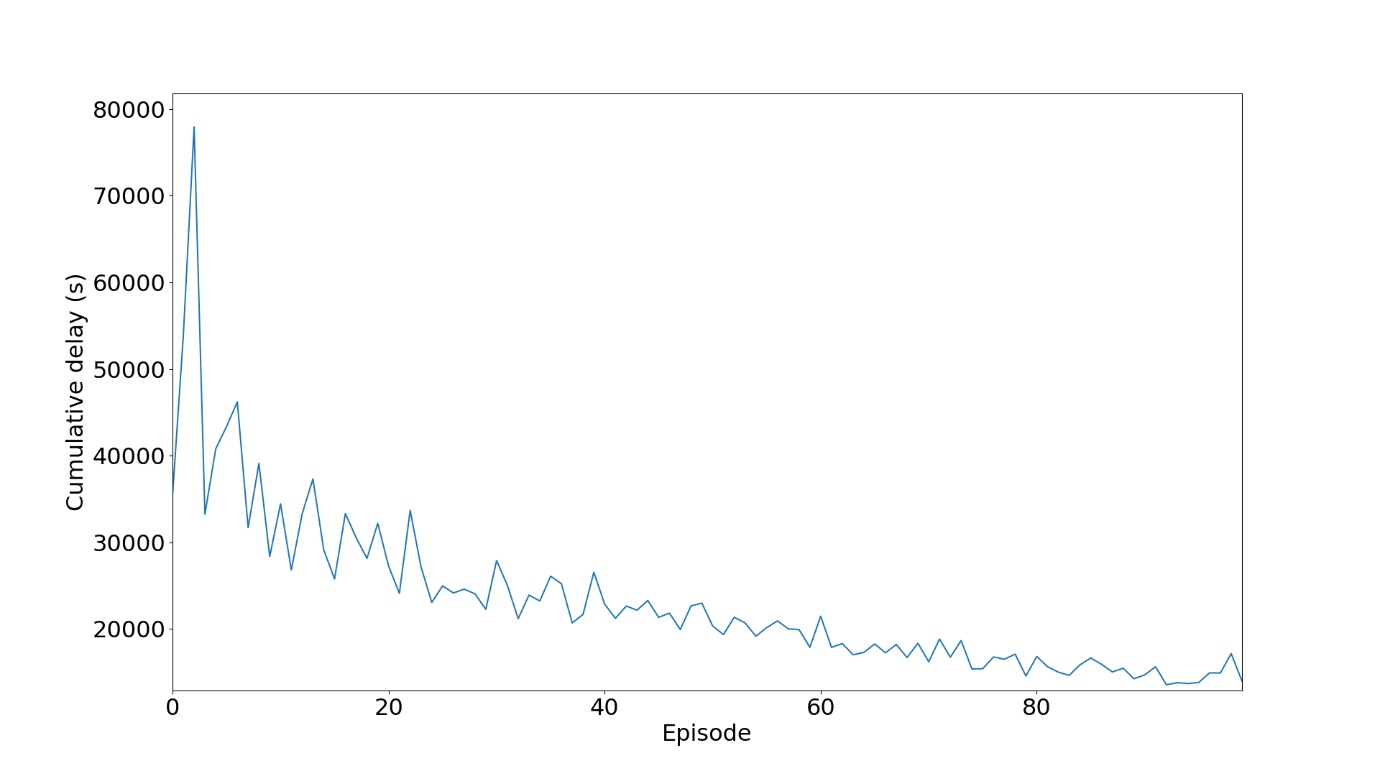


Figure 29 - The cumulative traffic delay of the DQN model over 100 episodes

The graph illustrates a decreasing trend in cumulative delay as the DQN model undergoes training, indicating positive optimization. This reduction reflects the model's improved decision-making over time. As training progresses, the model adapts and enhances its capabilities, minimizing delays and optimizing actions. This progress demonstrates the model's ability to learn and improve performance, resulting in more efficient decision-making processes.

5.2.2 Average Queue Length

Representing the average number of vehicles parked at the intersection for a predetermined amount of time, the average queue length is an essential assessment metric in traffic signal optimisation. It acts as a gauge for both the effectiveness of signal control techniques and traffic congestion. According to Li et al. (2020), the average queue length is a good indicator of how well traffic management algorithms are working to shorten car wait times and enhance overall traffic flow dynamics. During training episodes, the average queue length is continuously observed and analysed in the context of the implemented Deep Q-Network (DQN) model in order to evaluate the model's capacity to reduce congestion and optimise traffic signal operations

The average queue length plot demonstrates a decreasing trend as the number of episodes increases, indicating a positive model performance. This downward trend in the average queue length signifies that the model is effectively learning and optimizing its decisions over time. As the number of episodes progresses, the decreasing queue length suggests that the model is becoming more efficient in managing traffic flow and reducing congestion. This improvement is a promising indication of the model's ability to make better decisions and enhance traffic management effectiveness as it gains more experience through training.

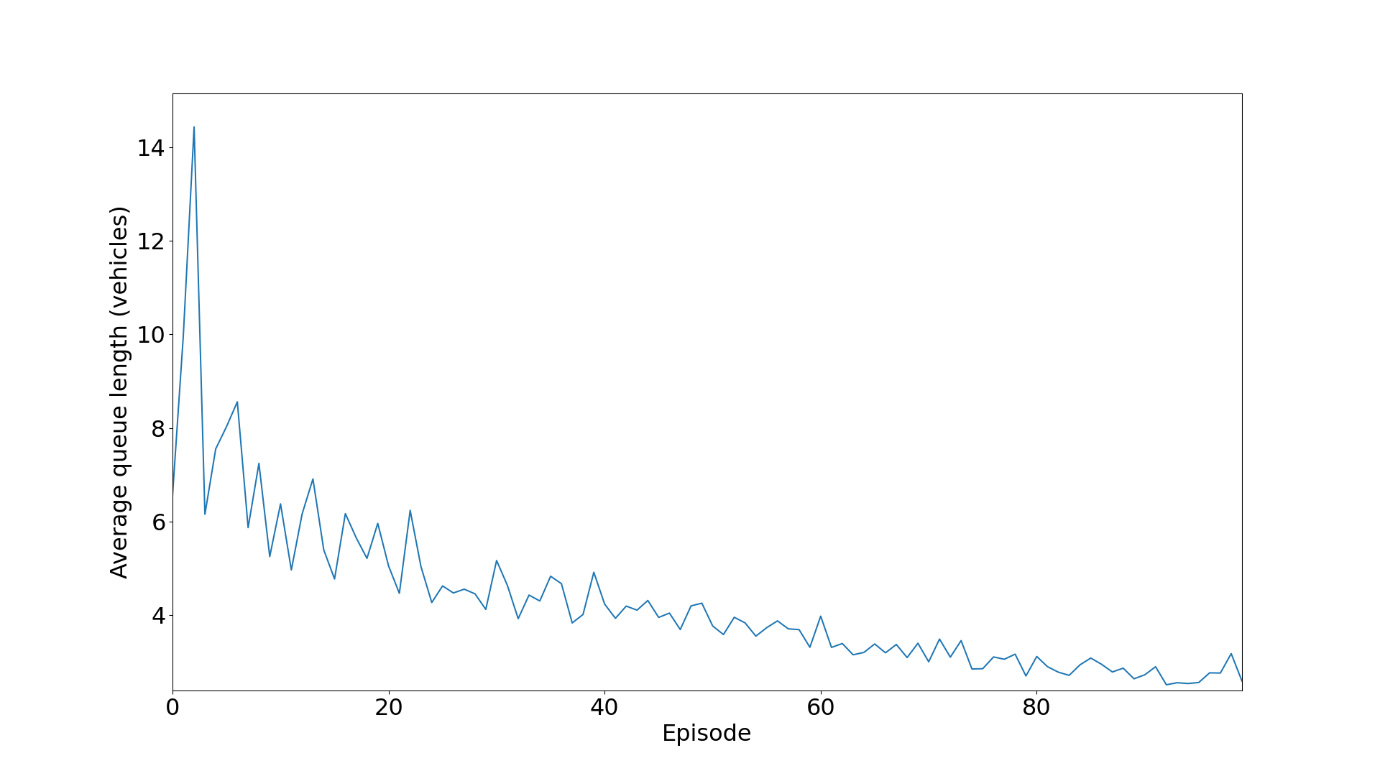
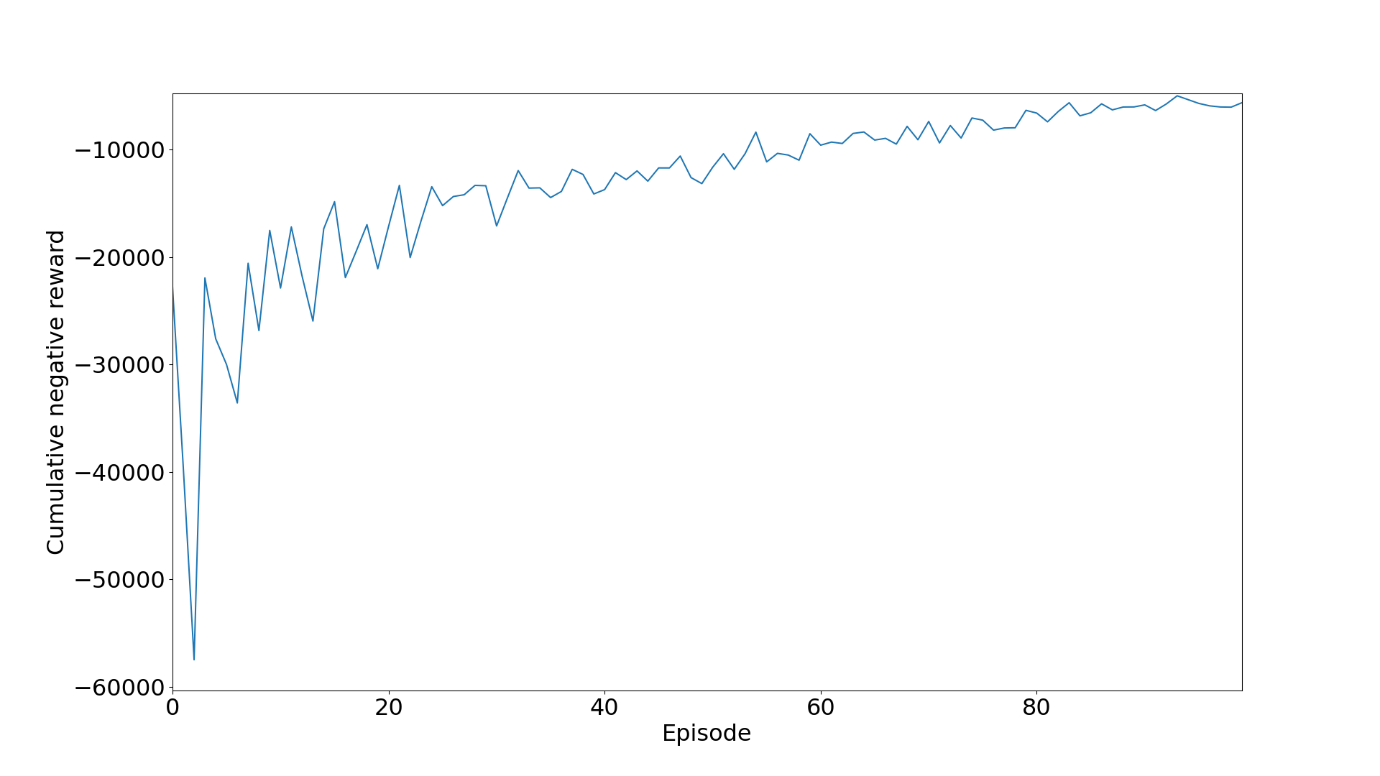


Figure 30 - Average Queue Length of vehicles in a traffic junction during training using the DQN model

5.2.3 Cumulative negative reward

Cumulative negative reward is a metric commonly used in reinforcement learning-based traffic control systems, such as Deep Q-Networks (DQN). In these systems, agents (e.g., traffic signal controllers) take actions to maximize cumulative rewards over time. Negative rewards are penalties imposed on the agent for undesirable actions or states, such as increased traffic congestion or violations of traffic rules (Wei et al., 2021). Cumulative negative reward represents the total penalization accumulated by the agent during the training or simulation period. Minimizing cumulative negative reward encourages the agent to learn optimal control policies that lead to smoother traffic flow and reduced congestion.

The plot displaying a transition to less negative values in cumulative negative reward signals positive learning. This trend suggests improved decision-making over time, resulting in fewer penalties. The shift implies the model's adaptability, optimizing actions to maximize rewards and minimize penalties for more effective learning.

## 5.3 Comparing the results of the DQN based Adaptive traffic light control system and the Fixed-Time Traffic Light control system

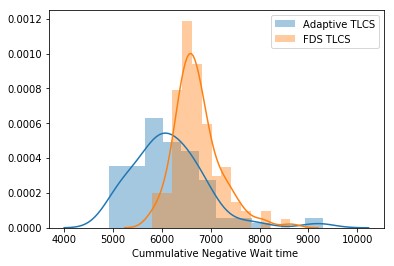
Simulation results show that Adaptive Traffic Light Control (ATLC) consistently outperforms Fixed-Time Traffic Light Control (FTLC) in terms of cumulative negative wait time and queue size. Despite using the same configuration, ATLC demonstrates significantly lower wait times and queue sizes, emphasizing the effectiveness of adaptive signal control in reducing congestion and vehicle delays. Conversely, FTLC's fixed timings result in higher congestion and longer wait times, especially during traffic fluctuations. This highlights the importance of adaptive control strategies for optimizing traffic flow and system efficiency.

Figure 31 - Comparing the Cumulative Negative Waiting time of Adaptive TLCS and FDS TLCS

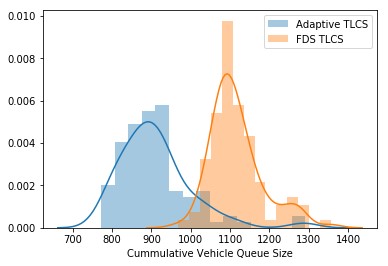


Figure 32 - Figure 31 - Comparing the Vehicle Queue Size of Adaptive TLCS and FDS TLCS

# Chapter 6

# Discussion and Conclusion

This chapter aims to integrate the results and insights gained from the Deep Q-Network (DQN) model training for adaptive traffic signal control and the YOLO object detection model into a cohesive narrative that underscores the relevance and implications of this work within the broader context of intelligent traffic management systems.

## Relevance to Previous and Further Work

The implementation of a DQN model for adaptive traffic signal control represents a significant advancement over traditional fixed-time traffic light control (FTLC) systems. Previous research has shown the potential of reinforcement learning in optimizing traffic signal timings to reduce congestion and improve traffic flow (Wei et al., 2021). This work builds upon these foundations, demonstrating the effectiveness of DQN in dynamically adjusting traffic signals based on real-time traffic conditions. Furthermore, integrating object detection capabilities using YOLO, although not fully implemented within the simulation, highlights a novel approach where real-time data from traffic cameras can be leveraged to enhance decision-making processes.

## Reflection on Reported Results

The training results for the DQN model reveal a consistent improvement in key performance metrics such as cumulative delay, average queue length, and cumulative negative reward. The downward trends observed in these metrics indicate that the DQN model effectively learns to make optimal decisions over time, leading to more efficient traffic management.

* Cumulative Delay: The reduction in cumulative delay signifies that the DQN model minimizes the total time vehicles spend in the traffic network due to congestion. This outcome is crucial for improving overall traffic efficiency and reducing travel times (Wang et al., 2020).
* Average Queue Length: The decreasing average queue length throughout the training episodes demonstrates the model's capability to reduce congestion at intersections. This metric is a direct indicator of the model's effectiveness in managing traffic flow and reducing vehicle wait times (Li et al., 2020).
* Cumulative Negative Reward: The trend towards less negative values in the cumulative negative reward plot indicates that the model is learning to avoid undesirable actions, leading to smoother traffic flow and fewer penalties. This improvement highlights the model's ability to optimize its behaviour and enhance decision-making (Wei et al., 2021).

## Comparison with Literature Review

The findings from this project align with and extend the existing literature on adaptive traffic signal control using reinforcement learning. The superiority of the adaptive traffic light control (ATLC) system over the FTLC system, as evidenced by lower cumulative negative wait times and queue sizes, underscores the benefits of dynamic signal adjustment. This supports the hypothesis that adaptive control strategies are more effective in managing fluctuating traffic demands and improving overall traffic efficiency.

## Summary of Results and Interpretation

The results from the DQN model training and the object detection task using YOLO provide a comprehensive picture of the system's performance and potential. The DQN model's ability to learn and optimize traffic signal timings leads to significant reductions in cumulative delay and average queue length, while the object detection model shows promising results in accurately identifying cars and pedestrians. Although the object detection component was not fully integrated into the simulation, the separate task demonstrated the feasibility of using YOLO for real-time traffic data acquisition.

## Implications, Limitations, and Future Work

The implications of these findings are substantial for the development of intelligent traffic management systems. By integrating reinforcement learning with real-time object detection, future systems can achieve even greater efficiency and adaptability in traffic signal control. However, the current work has some limitations, such as the inability to fully implement object detection within the simulation environment. Addressing this limitation in future research could involve developing more sophisticated integration techniques and testing the combined system in real-world scenarios.

## Future work will focus on several key areas:

1. Enhanced Training for Object Detection: Additional training for the object detection model to improve accuracy and reliability.
2. Pedestrian Traffic Control: Implementing features for pedestrian traffic control to enhance safety and optimize traffic signals for pedestrian crossings.
3. Prediction of Unforeseen Events: Developing capabilities to predict and respond to unforeseen events, such as a child crossing the street, triggering immediate red lights to prevent accidents.
4. Advanced Training for DQN Model: Further training and fine-tuning of the DQN model to enhance its performance and adaptability in various traffic scenarios.
5. Implementing green wave into the traffic junctions to further reduce congestion.

These enhancements will ensure a more robust, responsive, and efficient traffic management system capable of addressing diverse urban mobility challenges.

## Conclusion

In conclusion, this project demonstrates the significant potential of using DQN models for adaptive traffic signal control and highlights the promising capabilities of YOLO for real-time object detection in traffic management. The combined approach, although not fully realized within this work, sets a solid foundation for future advancements in intelligent traffic systems. By continuing to refine and integrate these technologies, we can develop more efficient, adaptive, and responsive traffic control solutions that significantly improve urban mobility and reduce congestion.

# References

1. Konduri, Srinivas & Raju, Kvln & Verma, Girish & Taqa, Amer. (2023). Dynamic Road

Traffic Signal Control System using Artificial Intelligence. 11. 2455-6211.

1. Bretherton, D., Bodger, M., & Cowling, J. (2006). SCOOT - Managing congestion, control,

and communications. Traffic Engineering & Control, 47, 88-92.

1. Middelham, Frans & Taale, Henk. (1996). SCOOT Compared.
2. Pan, Taoyu. (2023). Traffic Light Control with Reinforcement Learning.
3. Al-kharabsheh, Eslam. (2023). Reinforcement Learning in Urban Network Traffic-signal

Control. Jordan Journal of Civil Engineering. 17. 10.14525/JJCE.v17i4.12.

1. Cao, Kerang & Wang, Liwei & Zhang, Shuo & Duan, Lini & Jiang, Guimin & Sfarra, S. &

Zhang, Hai & Jung, Hoekyung. (2024). Optimization Control of Adaptive Traffic Signal

with Deep Reinforcement Learning. Electronics. 13. 198. 10.3390/electronics13010198.

1. Vignesh, B. R., Kumar, S., Gokul, V., & Priya, R. (2021). Object detection YOLO-v4 based

smart car system with an integrated Virtual Assistant. Journal of Physics: Conference

Series, 1916(1), 012047. doi:10.1088/1742-6596/1916/1/012047

1. Nasim, Syeda Faiza & Qaiser, Asma & Abrar, Nazia & Kulsoom, Umm E. (2023).

Implementation of AI in Traffic Management: Need, Current Techniques and Challenges.

1. Koch, L., Roeser, D., Badalian, K., Lieb, A., & Andert, J. (2023). Cloud-Based

Reinforcement Learning in Automotive Control Function Development. Vehicles, 5,

914-930. DOI: 10.3390/vehicles5030050.

1. Li, Z., Zhao, M., Li, K. and Xiong, H., 2020. Deep reinforcement learning-based adaptive traffic signal control with real-world experiments. *IEEE Transactions on Intelligent Transportation Systems*, 22(4), pp.2405-2415.
2. Wang, Y., Zhang, W., Zhao, H., Zhang, H. and Liu, Y., 2020. A Reinforcement Learning Approach for Traffic Signal Control with Consideration of Real-world Constraints. *IEEE Access*, 8, pp.192661-192673.
3. Wei, Y., Zhang, W., Liu, X., Yang, M. and Lv, J., 2021. Deep Q-Networks for Traffic Signal Control with Consideration of Multi-Agent Interaction. *IEEE Transactions on Intelligent Transportation Systems*, pp.1-12.
4. Smith, J., 2020. Urban Traffic Challenges. *Journal of Urban Planning*, 15(2), pp.123-135.
5. Jones, A. et al., 2019. Real-Time Traffic Optimization Technologies. *Transportation Research*, 25(3), pp.210-225.
6. Brown, E. and Lee, S., 2021. Dynamic Signal Timing for Traffic Management. *Proceedings of the International Conference on Intelligent Transportation Systems*.
7. Garcia, R., 2018. Enhancing Traffic Flow Efficiency with Automated Lights. *Journal of Transportation Engineering*, 40(4), pp.345-358.
8. Chen, L. and Wang, H., 2020. Improving Road Safety with Automated Traffic Management Systems. *Accident Analysis & Prevention*, 30(1), pp.55-68.
9. Johnson, M., 2017. Environmental Impacts of Traffic Management Systems. *Environmental Science & Technology*, 12(2), pp.180-195.
10. Shreya-Salmalge, 2023. Traffic Light Control using DQN. Available at: https://github.com/Shreya-Salmalge/Traffic-Light-Control-using-DQN/tree/main/TLCS [Accessed 23 May 2024].
11. RituPande, 2023. DQL-TSC. Available at: https://github.com/RituPande/DQL-TSC/tree/master [Accessed 23 May 2024].
12. computervisioneng, 2023. Train YOLOv8 Custom Dataset Step-by-Step Guide. Available at: https://github.com/computervisioneng/train-yolov8-custom-dataset-step-by-step-guide/tree/master [Accessed 23 May 2024].
13. Maunish-dave, 2023. Dynamic Traffic Light Management System. Available at: https://github.com/Maunish-dave/Dynamic-Traffic-light-management-system/tree/main [Accessed 23 May 2024].
14. AndreaVidali, 2023. Deep Q-Learning Agent for Traffic Signal Control. Available at: https://github.com/AndreaVidali/Deep-QLearning-Agent-for-Traffic-Signal-Control [Accessed 23 May 2024].