Smart Traffic Light System with Multi-Head Attention Deep Q-Learning for AI-Driven Vehicle Prioritization

BY

Kavina Shree A/P Ganason

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude and appreciation to my supervisors, **Dr.Rahmad Sadli**, who has given me this bright opportunity to engage in this Smart Traffic Light System with Multi-Head Attention Deep Q-Learning for AI-Driven Vehicle Prioritization project. His suggestions and motivation have been an instrumental in guiding this project into successful completion. Other than that, I am happy to thank all the lecturer and staff of UTAR for providing the knowledge, resources and their suggestions improved both the technical quality and presentation of my Final Year Project 2.

Besides that, I would also like to offer my sincere appreciation to my friends and family for their unconditional support, patience and motivation. Their kind heartedness, understanding and belief in me have bestowed the power of heart and will to continue, especially through problematic and difficult times. I am sensorily grateful for all their love and support which have been crucial in helping me to complete the FYP2 successfully.

ABSTRACT

Heavy traffic in populated areas leads to significant breakdowns in the performance of traditional traffic control systems. AI and adaptive traffic control systems are used to minimize these challenges. This project investigates the development of an advanced Smart Traffic system with Multi-head Attention Deep Q-Learning for AI driven vehicle prioritization. The goal of this project is to offer a simulation-based traffic control system that adjusts real-time traffic flow and prioritizes emergency vehicles. The system uses real-time data to make effective decisions. Emergency vehicle prioritization is a key component to ensure that the vehicles are given instant clearance to proceed through intersections.

In this era, many individuals face difficulties reaching their destinations on time due to traffic delays. This results in irregular traffic distribution, where one lane is packed with vehicles while another remains empty. Moreover, current traffic management systems are often too slow at adapting the signal conditions. In this project, traffic conditions will be monitored by detecting the number of vehicles in each lane. Priority is given to lanes with the highest vehicle count. This system will continuously make real-time adjustments based on traffic simulations.

In addition, the vehicle priority management system can detect emergency vehicles and override regular traffic control rules to allow them to pass first. To address the traffic congestion problems featured in this project, we are developing an advanced Smart Traffic system with an AI driven solution through this final year project to improve traffic signal control and reduce delays caused by irregular traffic flow. By merging emergency vehicle prioritization and real-time vehicle detection, this system aims to enhance traffic signal timing and improve overall road efficiency in high traffic urban zones.

Area of Study: Traffic Management Systems and Artificial Intelligence (AI), Deep Reinforcement Learning, Intelligent Transport Systems, Simulation-Based Modelling

Keywords: Multi-head Attention, Deep Q-Learning, Emergency Vehicle Prioritization, Traffic Management, Traffic Congestion, Simulation-based.

TABLE OF CONTENTS

TITLE PAGE **I**

COPYRIGHT STATEMENT **II**

ACKNOWLEDGEMENTS **III**

ABSTRACT **IV**

TABLE OF CONTENTS **V**

LIST OF FIGURES **X**

LIST OF TABLES **XII**

LIST OF SYMBOLS **XIII**

LIST OF ABBREVIATIONS **XV**

CHAPTER 1 INTRODUCTION **1**

1.2 Problem Statement 1

1.3 Motivation 4

1.4 Research Objectives 4

1.5 Project Scope 5

1.6 Impact, Significance and Contributions 6

1.7 Background Information 7

CHAPTER 2 LITERATURE REVIEWS **11**

2.1 Previous works on Deep Learning 11
2.1.1 Control of Emergency Vehicles with Deep Q-Learning [11] 11
2.1.2 Deep Q-network-based traffic signal control models [12] 13
2.1.3 Adaptive Traffic Signal Control with Deep Recurrent Q-learning [13]. 14

2.2 Limitation of Previous Studies 16

2.3 Proposed Solutions 20

2.4 Review of Technologies 23
2.4.1 Hardware Platform 23
2.4.2 Firmware / Operating System 23
2.4.3 Database 23
2.4.4 Programming Language 24

CHAPTER 3 SYSTEM METHODOLOGY AND APPROACH 25

3.1 Overview of Methodology 25

3.2 Reinforcement Learning Problem Definition 26

3.3 System Design 26
3.3.1 High-Level Architecture Diagram 27
3.3.2 Agent Environment Integration 29
3.3.3 Use Case Diagram and Description 30
3.3.4 Activity Diagram 31
3.3.4.1 Activity Diagram of the MHA-DQN Agent Training Process 31
3.3.4.2 Activity Diagram of the Vanilla Training Process 34
3.3.4.3 Activity Diagram of the Baseline Training Process 36

3.4 State Space Definition 37
3.4.1 Spatial State Vector 37
3.4.2 Global intersection features 38
3.4.4 Full spatial state tensor 39
3.4.5 Temporal State Tensor 40

3.5 Reward Function Design 40
3.5.1 Throughput Reward 41
3.5.2 Speed Reward 42
3.5.3 Flow Reward 42
3.5.4 Balance Reward (Fairness) 43
3.5.5 Queue Penalty 43
3.5.6 Waiting Penalty 43
3.5.7 Emergency Reward 44
3.5.8 Phase Switch Penalty 45
3.5.9 Total Reward 45

3.6 Multi-Head Attention Deep Q-Network Architecture 46
3.6.1 Spatial Attention Encoder 47
3.6.2 Scaled Dot-Product Attention 47
3.6.3 Temporal Attention Encoder 48
3.6.4 Q-value output layer 49

3.7 DQN with Prioritized Experience Replay 50
3.7.1 Double DQN Target 50
3.7.2 Prioritized Experience Replay 51
3.7.3 Action selection policy 52

3.8 Emergency Vehicle Preemption 52

3.8.1: Emergency detection condition 52
3.8.2 Emergency phase mapping 53

3.9 Smart Queue Switch 54

3.10 Timeline Diagram 55

CHAPTER 4 SYSTEM DESIGN 56

4.1 System Block Diagram 56

4.2 System Components Specifications 57
4.2.1 Neural Network Architecture Specifications 57
4.2.2 Vanilla DQN Specifications 58
4.2.3 Reinforcement Learning Hyperparameters 59
4.2.4 Reward Function Components 60
4.2.5 Action Duration Discretization 60
4.2.6 Emergency Vehicle Spawner Specifications 61
4.2.7 State Feature 61

4.3 Software Components Design 62
4.3.1 State Representation Design 62
4.3.2 Smart Action Selection Hierarchy 63

4.4 System Components Interaction Operations 64
4.4.1 Training Loop Interaction 64
4.4.2 Three-Agent Comparison Protocol 64

CHAPTER 5 SYSTEM IMPLEMENTATION 65

5.1 System Requirements 65
5.1.1 Hardware Setup 65

5.2 Software Setup 66
5.2.1 Simulation in Visual Studio Code which mainly used SUMO features 68

5.3 Settings and Configuration 69
5.3.1 Simulation Network Setup 69
5.3.2 Vehicle and Traffic Setup 69

5.4 System Operation 71
5.4.1 Launching the Training 71
5.4.2 SUMO Simulation Environment 71
5.4.2.1 SUMO Environment Setup 71
5.4.2.2 Integration Vehicle Handling 72
5.4.2.3 SUMO Environment with Vehicles 73
5.4.3 Per-Episode Console Output 73
5.4.4 Training Progress Plots 74
5.4.5 Model Checkpoints and Data Export 75

5.5 Data Logging and Visualization 77
5.5.1 Saved Artifacts 77

5.6 Implementation of Issues and Challenges 77

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION 79

6.1 System Testing and Performance Metrics 79
6.1.1 What Each Controller Does 79
6.1.2 Performance Metrics 80
6.1.3 How Emergency Prioritisation Works 81
6.1.4 Training Convergence Analysis 81
6.1.5 MHA-DQN vs Vanilla DQN Comparison 82

6.2 Testing Setup and Result 83
6.2.1 Testing Environment 83
6.2.2 Test Scenarios 83
6.2.3 Scenario S1 Results — 3 Emergency Vehicles per Episode 84
6.2.4 Scenario S2 Results — 5 Emergency Vehicles per Episode 86
6.2.5 Scenario S3 Results — 8 Emergency Vehicles per Episode 87
6.2.6 Cross-Scenario Comparison: S1, S2, and S3 89
6.2.7 Overall Performance Comparison Across All Scenarios 90
6.2.8 Overall Improvement Summary 90

6.3 Project Challenges 91
6.3.1 Emergency Waiting Time Varied Unexpectedly 91
6.3.2 Ensuring stability for 600 episodes of training 91
6.3.3 Graphics Driver Issue with the Network Design Tool 91

6.4 Objectives Evaluation 92

6.5 Concluding Remark 93

CHAPTER 7 CONCLUSION AND RECOMMENDATION 95

7.1 Conclusion 95

7.2 Recommendation 97
7.2.1 Multi-Intersection Network Scaling 97
7.2.2 Proactive Emergency Vehicle Green Wave 97
7.2.3 Real Traffic Data instead of Simulation 97
7.2.4 Emergency Vehicle Type Differentiation 98

7.3 Summary of Objectives Achievement 98

REFERENCES 99

REFERENCES FOR IMAGE 101

APPENDIX **A-1**

Poster A-8

LIST OF FIGURES

Figure Number	Title	Page
Figure 1.2.1	Stress Due to High Traffic	2
Figure 1.2.2	Accident	2
Figure 1.2.3	Emergency Vehicle Stuck in Traffic	3
Figure 1.5.1	Traffic Congestion	9
Figure 1.5.2	Simulation-based Environment Traffic System	9
Figure 2.1.1	Ambulance Detection1	12
Figure 2.1.2	Ambulance Detection 2	12
Figure 2.1.3	Simulation Traffic Control 1	13
Figure 2.1.4	Simulation Traffic Control 2	14
Figure 2.1.5	Traffic light Detect Emergency Vehicle	15
Figure 2.1.6	Traffic light change based on simulation	15
Figure 2.2.1	Ambulance Simulation Environment	16
Figure 2.2.2	Non prioritized fire truck in traffic	17
Figure 2.2.3	Fire Truck in heavy traffic	18
Figure 2.2.4	High Cost	19
Figure 2.2.5	Panic Attack	20
Figure 2.3.1	Emergency Vehicle Prioritize	20
Figure 2.3.2	Priority level	21
Figure 2.3.3	Traffic Signal detect Fire Truck	21
Figure 2.3.4	Vehicle Type	22
Figure 2.3.5	Observe real-time traffic control	22
Figure 3.3.1	High-Level System Architecture	27
Figure 3.3.2	Reinforcement Learning Model Diagram	29
Figure 3.3.3	Use Case Diagram of the Smart Traffic Light System	30
Figure 3.3.4.1	Activity Diagram of the MHA-DQN Agent Training Process	32
Figure 3.3.4.2	Activity Diagram of the Vanilla DQN Agent Training Process	34

Figure 3.2.4.3	Activity Diagram of the Baseline Fixed-Time Controller	36
Figure 3.5	Reward Function Design Diagram	40
Figure 3.6	Multi-Head Attention Deep Q-Network Architecture	46
Figure 3.9	Smart Queue Switch	54
Figure 3.10	Timeline Diagram	55
Figure 4.1	Block Diagram	56
Figure 5.2.1	SUMO Simulation (App)	66
Figure 5.2.2	Visual Studio Code (App)	68
Figure 5.4.2.1	SUMO lane diagram	72
Figure 5.4.2.2	SUMO rou.xml diagram (define vehicle)	72
Figure 5.4.2.3	SUMO vehicles	73
Figure 5.4.3	Terminal per-episode output	74
Figure 5.4.4.1	Model Training Graph	75
Figure 5.4.4.2	Comparison Model Training Graph	75
Figure 5.4.5	File system view of saved models and JSON	76
Figure 6.2.3.1	3 emergency Vehicle Summary	85
Figure 6.2.4.1	5 emergency Vehicle Summary	86
Figure 6.2.5.1	8 emergency Vehicle Summary	88

LIST OF TABLES

Table Number	Title	Page
Table 3.6.1	Reward Function Component Summary	50
Table 4.2.1	MHA-DDQN Neural Network Specifications	58
Table 4.2.2	Vanilla DQN Specification	58
Table 4.2.3	Reinforcement Learning Hyperparameters	59
Table 4.2.4	Reward Function Components and Formulae	60
Table 4.2.5	Action space mapping	60
Table 4.2.6	Emergency Vehicle Spawner Specifications	61
Table 4.2.7	Spatial features	61
Table 5.1.1	Specification of Laptop	65
Table 5.3.2	Python CONFIG Dictionary Settings	70
Table 5.5.1	Saved Artifacts	77
Table 6.1	Performance metrics used across all three evaluation scenarios	80
Table 6.2	Test scenario definitions	83
Table 6.2.3	Mean evaluation results — Scenario S1 (3 emergency vehicles)	84
Table 6.2.4	Mean evaluation results — Scenario S2 (5 emergency vehicles)	86
Table 6.2.5	Mean evaluation results — Scenario S2 (8 emergency vehicles)	87
Table 6.2.6	RL agent performance across all three scenarios	89
Table 6.2.7	Testing Performance Comparison	90
Table 6.2.8	MHA-DDQN percentage improvement over Fixed-Time Baseline and Vanilla DQN across all scenarios	90
Table 6.3	Project objectives evaluation summary	92
Table 7.3	Summary of project objectives achievement	98

LIST OF SYMBOLS

$\boldsymbol{S}_t$	Spatial state tensor at time t, size 4×25
$\mathbf{T}_t$	Temporal state tensor at time t, size $10 \times 4 \times 25$
$\mathbf{s}_t^{\text{flat}}$	Flattened spatial state vector (100 dimensions) used by Vanilla DQN
q_d	Queue length (number of stopped vehicles) on approach $d \in$ {West,East,North,South}
o_d	Lane occupancy on approach d (0 to 1)
v_d	Mean vehicle speed on approach d (m/s)
c_d	Total vehicle count on approach d
Q_t	Total queue length across all directions at time t
C_t	Total number of vehicles in the network at time t
p_t	Current active traffic light phase index $\in \{0,1,2,3\}$
τ_t	Normalised elapsed time in current phase $= t_{\text{phase}}/T_{\max}$
$\mathbf{h}_{\text{green}}$	Normalised counts of the last $H = 10$ phases
a_t	Discrete action chosen by agent at time t, $a_t \in \{0,1,\dots,43\}$
$G_{\min}$	Minimum green duration (10 seconds)
$G_{\max}$	Maximum green duration (60 seconds)
ΔG	Green duration step size (5 seconds)
R_t	Immediate reward at time t (composite function)
$R_{\text{throughput}}$	Throughput reward component
R_{speed}	Speed reward component
R_{flow}	Flow reward component
R_{balance}	Balance reward (negative penalty for uneven queues)
$R_{\text{emergency}}$	Emergency vehicle reward (clipped to $[-15,+15]$)
P_{queue}	Queue penalty
P_{waiting}	Waiting penalty (non-emergency halted vehicles)
P_{switch}	Phase switch penalty
γ	Discount factor (0.99)
ϵ	Exploration rate for ϵ-greedy policy

λ	Epsilon decay rate (0.008)
α	Prioritised Experience Replay (PER) exponent (0.6)
β	PER importance-sampling exponent (0.4)
d_{model}	Embedding dimension (64)
h	Number of attention heads (8)
d_k	Dimension per attention head $= d_{\text{model}}/h = 8$
D_{preempt}	Emergency vehicle pre-emption distance (150 metres)
τ	Soft target update rate (polyak averaging)
B	Mini-batch size (64)
H	Temporal history length (10 steps)

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
DDQN	Double Deep Q-Network
DQL	Deep Q-Learning
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
FYP	Final Year Project
GPU	Graphics Processing Unit
JSON	JavaScript Object Notation
MDP	Markov Decision Process
MHA	Multi-Head Attention
MHA-DDQN	Multi-Head Attention Double Deep Q-Network
MHA-DQL	Multi-Head Attention Deep Q-Learning
MLP	Multi-Layer Perceptron
MSE	Mean Squared Error
PER	Prioritised Experience Replay
PNG	Portable Network Graphics
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
SUMO	Simulation of Urban Mobility
TD	Temporal Difference
TraCI	Traffic Control Interface
UTAR	Universiti Tunku Abdul Rahman

CHAPTER 1 Introduction

In this chapter, we mainly focus on developing a simulation–based traffic control system that can adjust with the real-time traffic conditions instead of relying on fixed timing and implementing vehicle priority which gives importance to emergency vehicles like fire trucks or ambulances to cross the traffic signal easily without any delay or obstacles.

1.2 Problem Statement

The management and control of city traffic lights are becoming a significant problem in many countries. The rising number of vehicles and the slow pace of highway development have led to traffic blockages and congestion problems especially in highly populated cities. It causes multiple problems such as longer travel times, environmental quality, quality of life, and road safety all of which are affected by traffic congestion [1]. In addition, delays due to traffic gridlock also indirectly affect productivity, efficiency, and energy loss for the public who are driving in the cities with heavy traffic jams.

There are several problems that cause this:

Complex traffic jam affects individuals' timely arrival.

- In this modern era, many individuals are constantly busy traveling in their daily lives. Most of them prefer to reach their destination on time or even earlier. But this has become very complicated due to unexpected gridlock that causes delays. As a result, most individuals face stress because of extended travel time. Other than that, the number of vehicles significantly increases during peak hours such as 12pm – 1pm and 4pm – 7pm which is common as countless people head to work or school simultaneously. Some traffic signals with poor timing cause longer waiting times. For example, when the traffic signal is not properly timed, vehicles are forced to wait longer even when the other side of the signal has no vehicle to pass.

Figure 1.2.1 Stress Due to High Traffic

Increase the risk of accidents

- Current traffic management systems can lead drivers to become confused about when to stop or go, leading to collisions. The unpredictable traffic signal changes might make drivers suddenly brake without a warning. This increases the risk of traffic accidents and affects public safety [2]. In addition, sudden traffic light changes can lead to accidents. For instance, when the traffic signal switches the light quickly without sufficient warning, drivers may not have enough time to respond or to stop safely, which could lead to unnecessary accidents.

Figure 1.2.2 Accident

Emergency Vehicles delayed

- The main challenges occur when traffic lights do not prioritize emergency vehicles, such as ambulances, fire trucks, or police vehicles, causing them to get stuck in regular traffic and waste valuable time. This late response is a major issue that can worsen the condition of emergency victims by delaying medical care or fire response time [3]. This does not only waste valuable time but can also result in the death of critical patients and cause fire trucks to be delayed in reaching the spot, leading to greater damage. Traditional traffic signals do not prioritize emergency vehicles. It is unsafe for the emergency vehicle to cross during a red-light signal. It can cause harm to the vehicles in other lanes who may end up in accidents due to sudden breaking in the middle of the road to allow the ambulance to pass.

Figure 1.2.3 Emergency Vehicle Stuck in traffic

Traditional traffic light control systems are not adaptive enough to adapt to real-time circumstances, and they do not give precedence to emergency vehicles such as ambulances, fire trucks and police cars or adapt to changing traffic conditions. Therefore, there is a need for smart traffic control system approaches using artificial intelligence, such as those utilizing reinforcement learning, like Multi-Head Attention Deep Q-Learning.

1.3 Motivation

The motivation of this project is to improve traffic control systems with advanced technology of Artificial Intelligence that prioritizes emergency vehicles. This project can control traffic signals automatically based on the real-time traffic data using a simulated environment. Since the system is computerized, it minimizes the mistakes that may happen with manual traffic. In this modern digital era, almost every family has at least one car or any vehicle to be used in their daily lives and this has caused the population of vehicles on the road to increase and cause traffic congestion. This project aims to address the three limitations which are **the timely arrival problem caused by high traffic**, **the high risk of accidents** and **the delay of emergency vehicles**.

Furthermore, the motivation for developing this traffic control system is to provide drivers with a smooth traffic flow with less-congested areas that can reduce the chances of accidents at intersections. And the automated emergency vehicle detection system will change the signal into green so it can avoid the delay of emergency vehicles. Reaching areas with less traffic congestion can reduce fuel usage compared to stop-and-go traffic. Efficient traffic controls can help to create a more sustainable, environmentally friendly city. By optimizing the traffic flow, it helps regular drivers to reach their destinations on time which reduces stress caused by long waiting times.

In general, the main goal of this improved Smart Traffic Control System using Multi-Head Attention Deep Q-Learning is to improve time saving and provide smart travel for the public. Multi-Head Attention Deep Q-Learning is chosen specifically because it improves decision-making by looking at different parts of the state, allowing for more accurate predictions and real-time traffic control. This is especially helpful when compared to traditional learning-through-reward learning methods, such as basic Deep Q-Learning, which might not understand complicated connections within the system. The system also aims to improve public safety by giving importance to emergency vehicles. Finally, it aims to help the smart city development with an intelligent traffic management system.

1.4 Research Objectives

The primary objective of this project is to plan, design and construct a simulation-based smart Artificial Intelligence traffic management system that would be able to improve traffic control, mainly at congested intersections. This system seeks to showcase how traffic

can be optimized using adaptive algorithms and digital detectors, minimizing real-world testing needs. There are five sub-objectives derived from the main objectives. First, construct an interactive simulation environment system. This system functions as a virtual testing platform for simulated-based environment traffic management strategies without relying on real-world systems. It will simulate vehicle flow at junctions using technological resources, enabling users to monitor traffic light functions under different situations.

Secondly, this system applies adaptive traffic management algorithms that rely on this strategy to employ input provided by virtual detectors to monitor the count as well as classification of vehicles in intersections. Based on this information, the system shall automatically adjust signal timings in real time, enhancing traffic efficiency while reducing congestion during peak hours. Thirdly, it will incorporate emergency vehicle recognition. This system shall include a component that detects emergency vehicles and dynamically alters traffic light sequences to clear a path for them. This helps simulate real-world scenarios where emergency response time is critical, while traffic systems must adjust to prioritize them.

Other than that, the platform offers manual control features. This option provides users or developers with complete control to simulate traffic system scenarios. They can adjust settings such as the duration of green or red signal, add traffic density, or simulate an obstruction. This makes the platform more flexible as well as useful for evaluating specific situations or research hypotheses.

In addition, the system presents graphical traffic movement visualizations which the simulation will include graphical tools in Sumo such as animated traffic lights and vehicle movement to make the system more interactive and intuitive. These visual aids assist drivers quickly recognizing the way traffic flow changes over time or based on different configurations.

1.5 Project Scope

This project aims to design and develop a simulated traffic signal control system which comes up with a well-organized and strong ways to manage traffic signal flows at intersection road, especially in the high population areas. By this simulation system it will be as a demonstration and testing platform for this smart traffic control system strategies. Other than that, it helps users to visualize traffic control systems in a virtual environment.

The major objective of this project is to design an interactive simulation that enables the user to test and evaluate different traffic control system algorithms in multiple traffic situations. The system will remove the physical tools or real-world traffic control testing needs; thus, it helps to reduce the cost, difficulty, and risks associated with traditional traffic system research. By including variable parameters such as signal timing, and vehicle priority control, viewers will be able to differentiate both primary and adaptive traffic control management scenarios.

This simulation system applies detectors to identify the vehicle flow and actively changes the signal dynamically, imitating current real-time traffic data. And it also contains emergency vehicle recognition, that enables the system to give priority to traffic signal changes and create a clear route for emergency vehicles such as ambulances and fire trucks. The simulation assists various intersections within a single environment, enabling more comprehensive scenario planning and comparative study. Additionally, it presents graphical visualizations of traffic flow and signal phase changes, making it more convenient for users to observe the flow and modify settings as needed.

This system includes manual traffic signal control features that allows direct configuration of traffic environment. Parameters such as traffic signal duration, vehicle flow, or obstructions can be modified to replicate varied conditions. This flexibility enhances the platform's usefulness for analysing specific situations, testing hypotheses, and evaluating traffic management strategies in controlled environments.

Furthermore, the system highlights public satisfaction by offering an instinctive, user-friendly interface. It simplifies the simulation configuration and operational process, supporting users from non-technical backgrounds to engage with the system effectively. This system enables exact visualization of traffic control flow and signal light changes, allowing public or readers to receive a stronger understanding of traffic light control logic together with vehicle prioritization and city mobility obstacles.

1.6 Impact, Significance and Contributions

The significance of this project relies on its ability to address one of the most crucial problems that encountered by the modern cities which is the issue of inefficient traffic management. Through implementing this Smart Traffic Light System with MHA-DQL, our project not only contributes to improve the queue length or reduce waiting time of every single

traffic lane but also have enhanced the sustainability for emergency vehicle to pass the signal easily. Readers and researchers who study this project will benefit by learning the insights into how advanced AI can be used to solve real-world traffic problems that affect everyone who drives. This system guarantees quicker clearance for emergency vehicles such as ambulances, fire trucks and so on that possibly saving countless of lives. Beyond social benefits, this project also has contributed on academically, by exploring the integration of reinforcement learning with attention mechanisms that offering future researchers a framework to build upon. Furthermore, through our project's solution it can influence urban planners to use AI-driven solutions for better city's traffic development. By reducing the traffic jam, lowering accident risks, and prioritizing critical services this project proves its long-term value. It inspires the readers to view technology not just as a tool or element for convenience but as a transformative force for societal good that helps to improve reduce problems sufficiently.

Moreover, it's worth the reader's attention because of the problem flow that every driver, and city resident faces in their daily life basis. Long queues, wasted fuel, increased stress and delay emergency vehicle responses are issues that directly affect the quality of life and cities peace. Through this project it shows how AI can dynamically adjust traffic signals in real time. It proves the readers with a solution that feels both practical and innovative. It shows that traffic control systems are not just a mechanical device but intelligent platform capable of learning, adapting, and evolving with society's needs. This makes the project highly desirable as it connect the technology with human condition. Lastly, the contribution is not only in academic exploration but also in inspiring the adoption of smarter and peaceful cities without traffic jams. Readers are encouraged to see the project as a forward-looking initiative that promotes safety, efficient and sustainability.

1.7 Background Information

Traffic in Urban areas is congested due to the large number of vehicles attempting to use the common traffic control infrastructure which have limited capacity. Traffic signals have been developed since 1912 and were implemented as signalling systems designed to control traffic flow at road intersections [4]. Traffic signal control is set up to manage traffic on streets and highways, particularly at road interactions, and to prevent accidents. A traffic light has three colours which are red, yellow and green. Every colour carries a sign. A red light means the road user must stop driving and not cross or proceed. The light typically lasts between 28 to 40 seconds. Secondly the yellow light indicates that the road user must be ready to stop their

vehicle. However, if the driver is too close to the stop line and cannot safely stop, they must continue moving. The yellow signal typically lasts for 2 to 5 seconds. Lastly, the green light indicates that the road user can continue their journey, there are no obstacles. The green signal typically lasts between 28 to 40 seconds [5]. Driving through a red light without reason is a traffic violation.

Moreover, the continuous growth in the number of cars has led to increased traffic congestion [6]. These challenges have become one of the most crucial problems faced by the cities today. As the number of vehicles continues to increase and the resources provided by the current traffic control infrastructure remain limited, the need for AI-based traffic signal systems is increasing [7]. This project conducts an in-depth analysis of traffic light functions and user conditions. Although traffic signal lights are very common and relatively easy to manufacture, they are crucial for ensuring safe driving conditions in certain areas. Traffic management systems using traditional methods alone are no longer enough sufficient to address unpredictable traffic flow smoothly, especially in areas with high population density. To mitigate the challenge, technologies such as artificial intelligence (AI) and deep learning must be applied to the traffic light management system to help the traffic flow smoothly [8]. The simulation-based traffic signal model is highly effective for modelling and analysing traffic scenarios to reduce gridlocks and improve the prioritization of public safety vehicles.

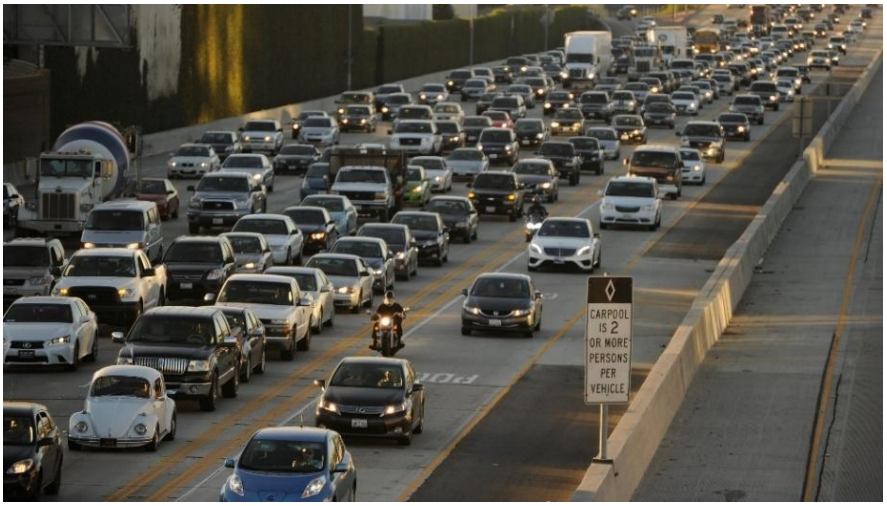


Figure 1.5.1 Traffic Congestion

This project mainly aims to develop an advanced Smart Traffic System that incorporates a multi-head attention mechanism in deep Q-learning to manage AI-driven vehicles successfully. This AI-driven approach will enable a system that is more adaptive and efficient compared to

traditional traffic systems. Multi-head attention is a powerful technique used in natural language processing and other AI fields and can be leveraged to improve decision-making in vehicle prioritization, enabling vehicles to pass to traffic signals more effectively.

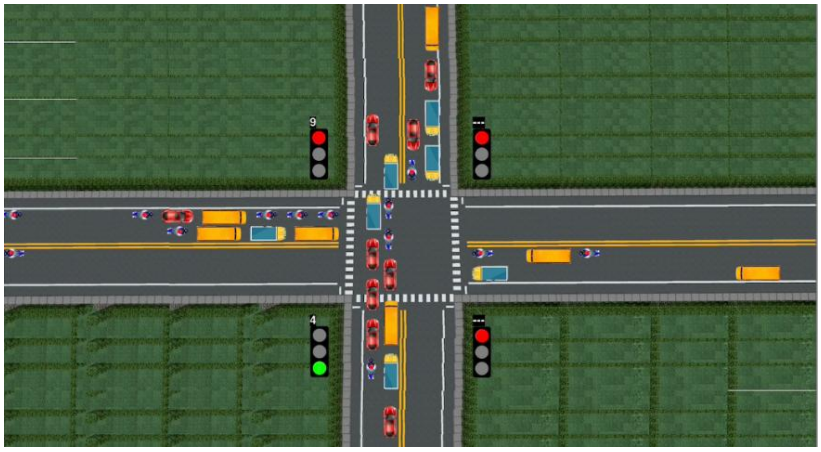


Figure 1.5.2 Simulation-based Environment Traffic System

Other than that, the system will mainly use a simulation-based environment which is used to represent certain parts of the real world by constructing a computer model to simulate it over time [9]. This simulation environment will be used to build a traffic control system that will implement a vehicle prioritization system to ensure a smooth flow for emergency vehicles [10]. Lastly, by combining the two systems these technologies will form the foundation of a modern AI-driven Smart Traffic Control System which will improve the effectiveness of the traffic management system. Through incorporating advanced AI and deep Q-learning techniques, this project aims to set a new standard for intelligent traffic control systems.

In addition, this system uses Multi Head Attention Deep Q-Learning which enables the artificial intelligence model to manage the difficult traffic situations like focusing on emergency vehicles and at the same time it considers the aspects of traffic jams and vehicle counts simultaneously. Not only that it allows the system to concentrate on many parts of the traffic road situation. The Deep Q-Learning is a type of reinforcement method which operates on learning the best traffic signal timing by tests and errors. The artificial intelligence system changes the signal phases with incentives provided for successful moves and helps reduce the delay periods or better flow movement. The use of multi-head focus within the Deep Q-Learning method which supports the improvement of the system's decision-making steps. This program can look at different traffic conditions on its own which allows it to make better situation-based choices from every junction

Lastly, MHA-DQL is better than other DRL variants because it is better in dealing with complicated traffic situations. Based on the current traffic system the DRL method usually has trouble with multilayered and dimensional issues since they handle points one by one. The updated method that MHA-DQL uses side-by-side processing and can look at different traffic elements simultaneously. It makes the traffic system more useful and efficient. Not only that MHA-DQL enables the system to prioritize the critical tasks such as giving priority to emergency vehicles which results in faster learning in changing environments. This helps the system shift the traffic situations faster than the other learning methods like DRL.

CHAPTER 2 Literature Reviews

2.1 Previous works on Deep Learning

This section reviews several articles related to emergency vehicle control using deep reinforcement learning techniques, namely Control of Emergency Vehicles with Deep Q-Learning [11], Deep Q-network-based traffic signal control models [12], and Adaptive Traffic Signal Control with Deep Recurrent Q-learning [13].

2.1.1 Control of Emergency Vehicles with Deep Q-Learning [11]

- Based on the studies "Control of Emergency Vehicles with Deep Q-Learning. It mainly discusses the significant delays faced by emergency vehicles such as fire trucks, ambulances and police vehicles due to heavy traffic congestion. These delays can result in serious impacts especially when considering critical care or responding to high-risk emergencies delivery. With the advancement of artificial intelligence and machine learning and reinforcement learning the study in [11] has developed a smart traffic system that dynamically adapts real-time situations with emergency vehicle detection using deep Q-learning. One of the purposes of the study in [11] is to develop a traffic control system that allows changing the signal based on real-time traffic conditions. Overall, the study in [11] is motivated by the urgent need to modernize emergency vehicle logistics in the face of growing urbanization and population density. It aims to bridge the gap between traditional emergency routing systems and intelligent transportation solutions using cutting-edge AI techniques [11]. By focusing on Deep Q-Learning, this research lays a foundation for the future of smart emergency mobility systems that are responsive, adaptive, and lifesaving.

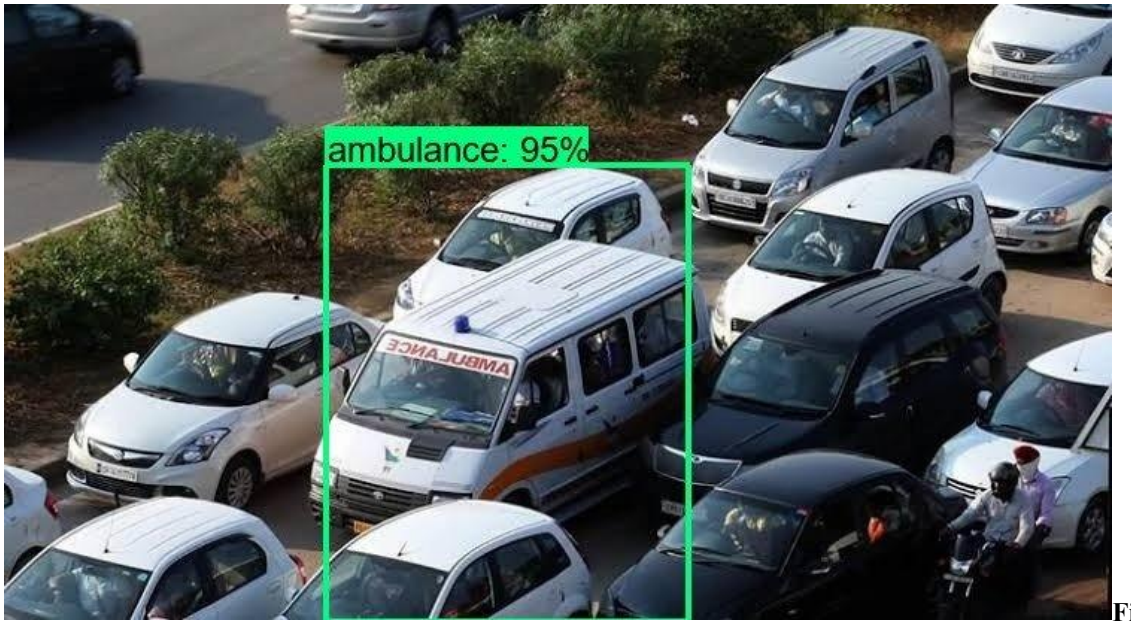


Figure 2.1.1 Ambulance Detection 1

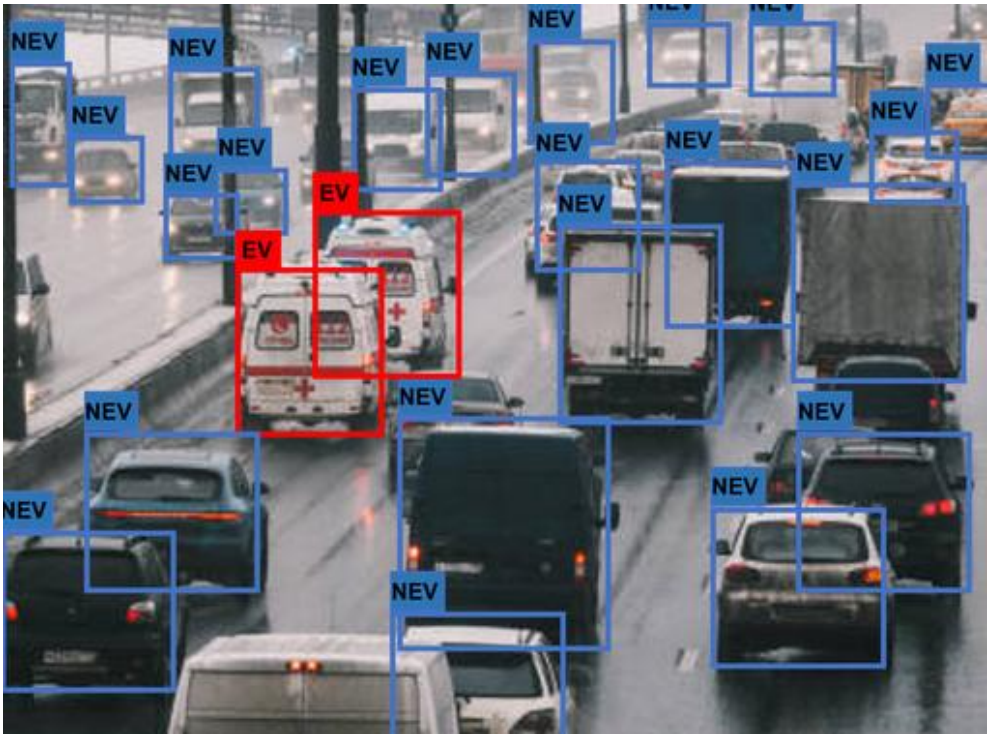


Figure 2.1.2 Ambulance Detection 2

2.1.2 Deep Q-network-based traffic signal control models [12]

- The system has employed pre-determined timing plans to allow a green traffic signal to the busy lane. In the study [12], they have used a simulation environment designed based on a real-world traffic signal layout for testing and developing the Deep Q-Learning model. It simulated the realistic vehicle behavior used to simulate road, intersections and traffic flow [12]. The simulation environment traffic control system examines the traffic condition and uses spatial distance as input for training the agent in choosing efficient routes. Multi-head attention does this by using several "perspectives" or "heads", so the model can catch different types of important traffic conduct simultaneously. Instead of treating all input equally, the attention mechanism gives higher importance to key patterns such as sudden traffic delays or consistently jammed roads

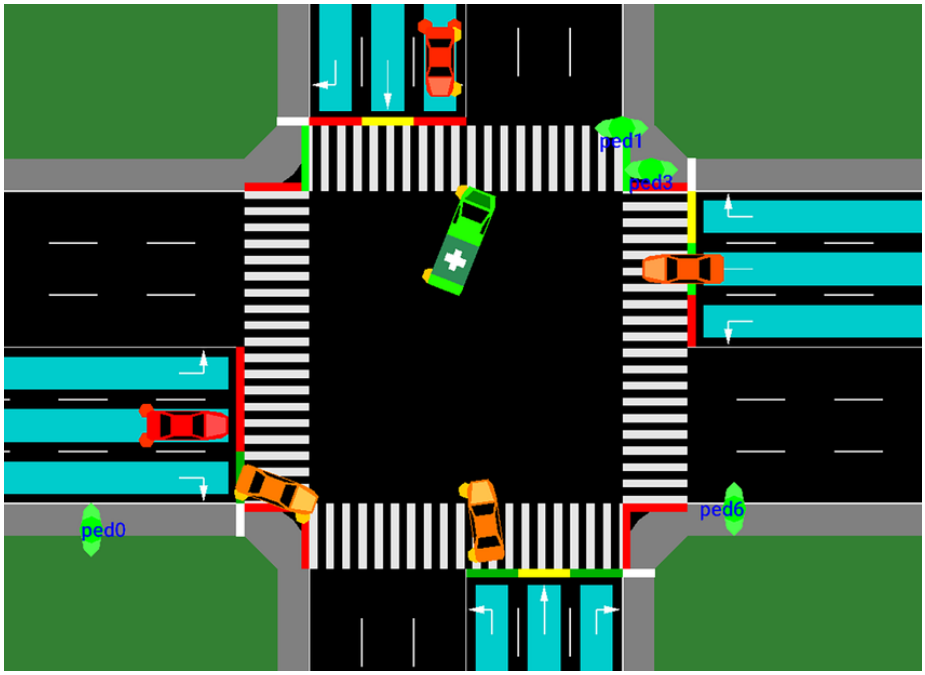


Figure 2.1.3 Simulation Traffic Control 1

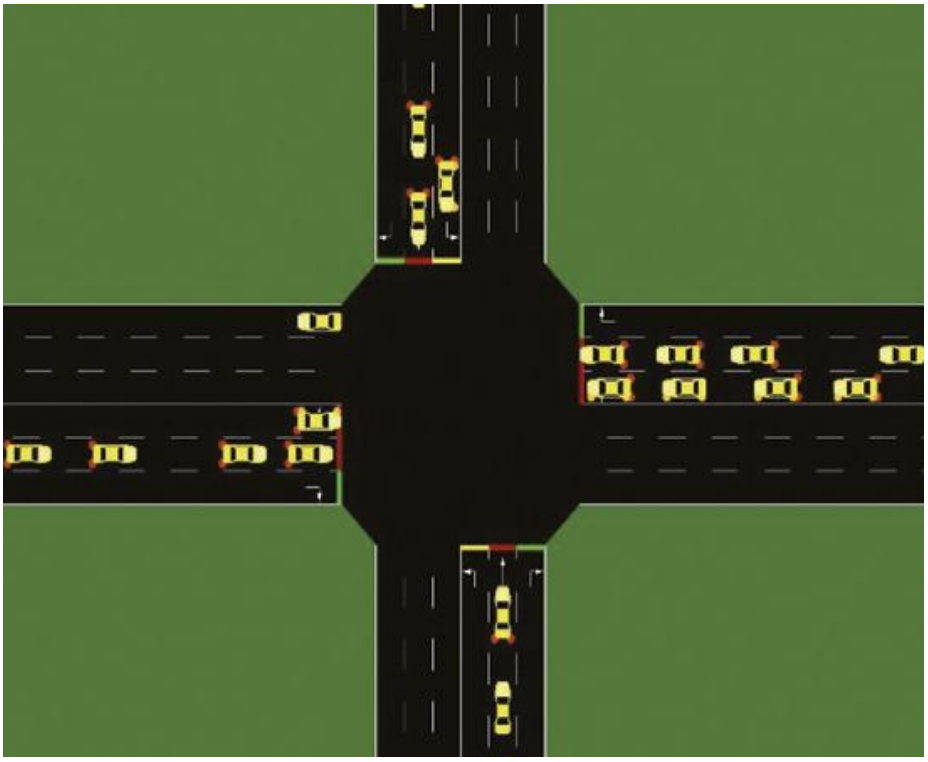


Figure 2.1.4 Simulation Traffic Control 2

2.1.3 Adaptive Traffic Signal Control with Deep Recurrent Q-learning [13].

- Based on this article, it has multiple benefits which are minimizing the delay time at road intersections and guarantees unobstructed path for the emergency vehicle to pass through the traffic signal [13]. In this study it emphasizes advanced smart traffic systems with **Deep Q-Learning** for **Artificial Intelligence-driven emergency vehicle prioritizing**, mainly focusses on emergency vehicles. It has the vehicle prioritization which the system applies **machine learning models** to prioritize emergency vehicles such as ambulances, fire trucks, and police vehicles that ensure faster response times by adjusting the traffic signal to green. Other than that, this article uses AI-driven optimization which improves the traffic control system's ability to **optimize** routes for emergency vehicles, balancing between traffic management and emergency vehicle prioritization [13]. The study employs a **simulated environment** based on real-world traffic signal layouts to simulate vehicle behaviours and simulate traffic flow, road, and intersection dynamics.

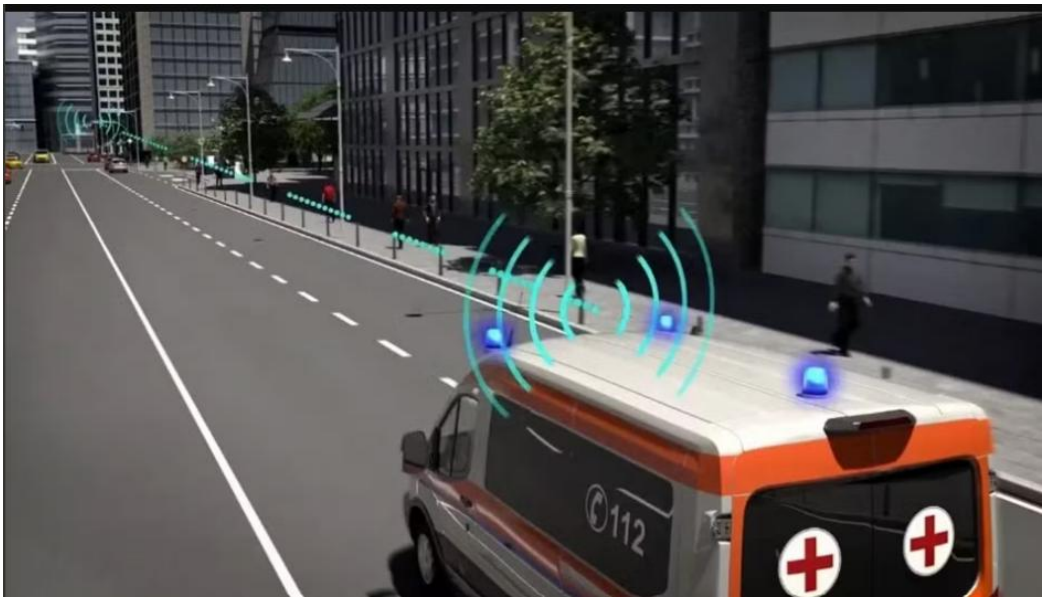


Figure 2.1.5 Traffic light Detect Emergency Vehicle

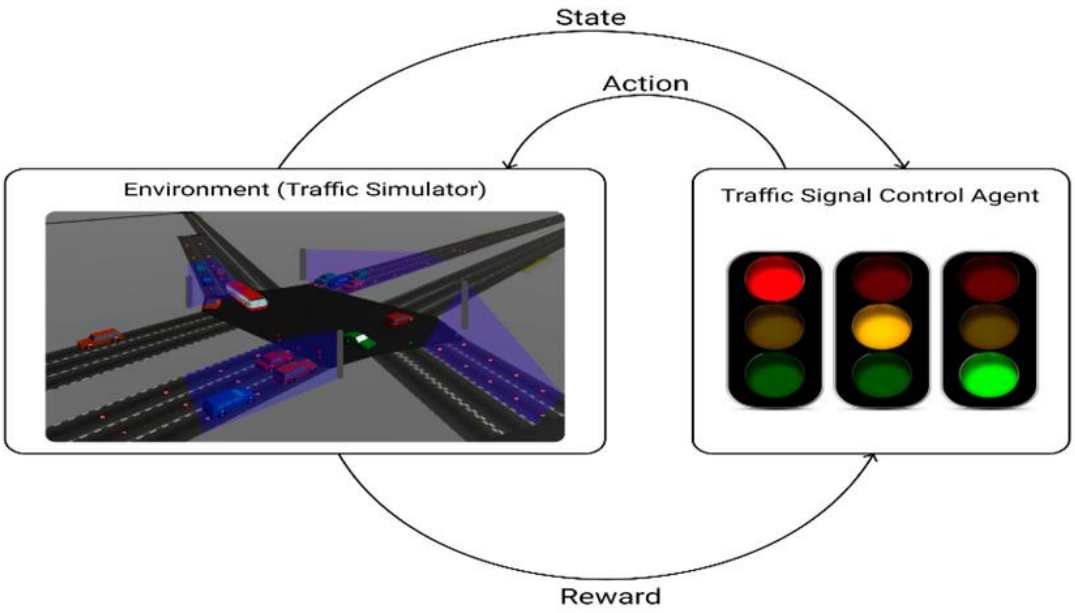


Figure 2.1.6 Traffic light change based on situation

2.2 Limitation of Previous Studies

Based on the previous studies, they did not cover some aspects which have led the article to certain limitations. By referring that three previous articles, we have identified 4 limitations which is **Limited Emergency Vehicle coverage** [11], **Lack of Emergency Vehicle Prioritization** [12], **High Computational Requirements**[12] and **Lack of Human Driver Behaviour Consideration** [13]

Lack of Emergency Vehicle coverage[11].

According to this research paper [11] it has mainly emphasized ambulance services as one type of emergency vehicle prioritization. However, it lacks importance on the priorities of other emergency services such as fire engines and police cars. Various kinds of emergency vehicles have different levels of urgency and treating them all the same may limit the system's ability to make better efficient and it can be one of the important life-saving decisions in emergency circumstances. Emergency Vehicle such as fire truck and police car are one of the important vehicles that often should be respond on time-sensitive circumstance. Based on this research's system it does not analyse the impact of other emergency vehicle delays rather than ambulance on overall emergency outcome. Lack of vehicle specification in this article may head to inefficiencies and reduce the effectiveness of the critical solution.

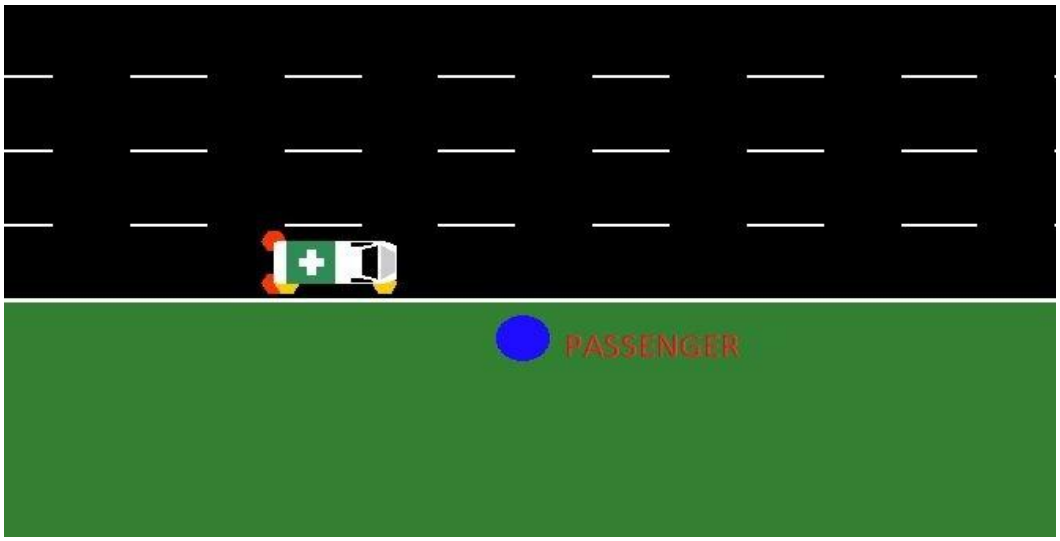


Figure 2.2.1 Ambulance Simulation Environment

Figure 2.2.2 Non prioritized fire truck in traffic

Lack of Emergency Vehicle Prioritization & High Computational Requirements [12].

Based on the research on Deep Q-network-based traffic signal control models. It fails to distinguish emergency vehicles from regular vehicles. As a result, emergency vehicles may not receive appropriate prioritization when traffic signals are controlled. Treating all regular vehicles the same may lead to critical delays in life-threatening situations such as fires, medical emergencies, or crimes in progress. This system can struggle to adapt in mixed traffic conditions where both emergencies and regular vehicles coexist. This limitation reduces the model's effectiveness since emergency vehicles need faster movement through traffic congestion to regular vehicles, particularly during high-risk or time-sensitive emergencies.

Figure 2.2.3 Fire Truck in heavy traffic

Furthermore, the article uses the system that designs to be dependent heavily on complex deep learning models like Deep Q-Networks with multi-head attention mechanisms. These models require significant computational power and memory which cause real-time applications to be challenged. Implementing these systems in big cities can be at a high cost and impractical without advanced infrastructure and regular maintenance to handle large data traffic. The approach may not be easily scalable across regions with differing traffic rules, road layouts, or emergency protocols

Figure 2.2.4 High Cost

Lack of Human Driver Behaviour Consideration [13]

The model believes that all drivers act reliably and obey traffic rules strictly, which is rarely the situation in actual real-world environments. Human drivers may become anxious, delay, block intersections, or respond unpredictably when emergency vehicles approach. Ignoring these variations in driver's behavior reduces the dependability and effectiveness of the framework, as it may overestimate how easily emergency vehicles can navigate traffic, leading to delays or unsafe path choices. The system does not account for hesitation or panic in drivers when encountering sirens or flashing lights, which may cause erratic movements. The model fails to simulate distracted or inexperienced drivers who may not notice emergency vehicles in time and Lack of integration with real-time visual data (e.g., cameras detecting human behavior patterns) reduce responsiveness to actual road conditions.

Figure 2.2.5 Panic Attack

2.3 Proposed Solutions

To overcome the limitations which were identified in the previous studies, there are some improvements that could be suggested. Firstly, the system needs to enhance emergency vehicle prioritization by including several types of emergency services, such as ambulances, police cars and fire trucks. A classification mechanism can be established to identify each type of emergency vehicle based on urgency level and checking that each emergency vehicle has been prioritized according to its specific operational requirement. Enable multi-agency coordination by synchronizing traffic signals with emergency dispatch systems.

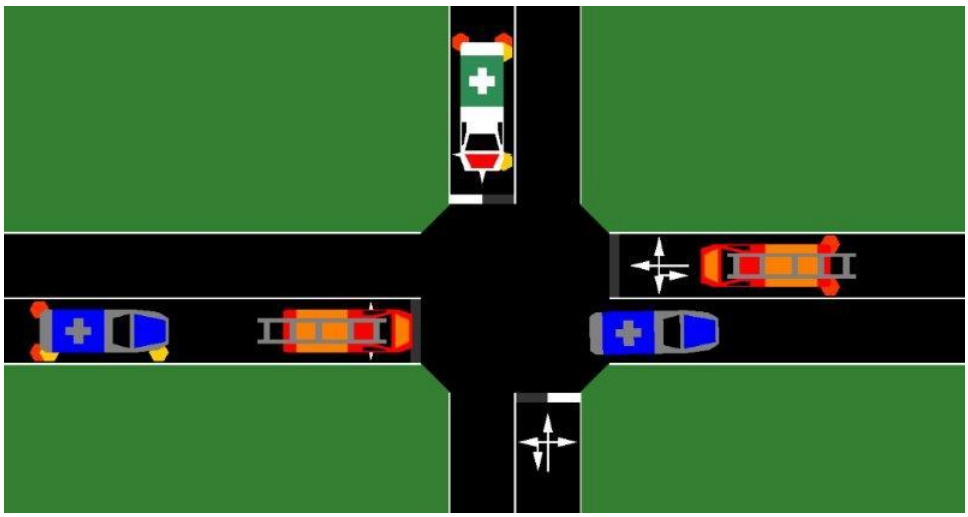


Figure 2.3.1 Emergency Vehicle Prioritize

Ambulance	Fire Service	Police	Normal Vechile
Priority : 1	Priority : 2	Priority : 3	Priority : 4
			

Figure 2.3.2 Priority level

Furthermore, to update the emergency vehicle prioritization, the traffic control system must use adaptive prioritization technique. By implementing these methods, it can analyse the real-time traffic record and emergency vehicle information to modify the traffic signals rapidly, thereby offering faster and less risky clearance through crowded roads. Implement predictive modelling to forecast traffic flow and pre-emptively adjust signal timing before congestion builds. Circumstantial factors like the intensity of the emergency and time dependency should also be believed to make more informed decision. Allow override functionality where emergency control centres can intervene in real time if needed

Figure 2.3.3 Traffic Signal detect Fire Truck

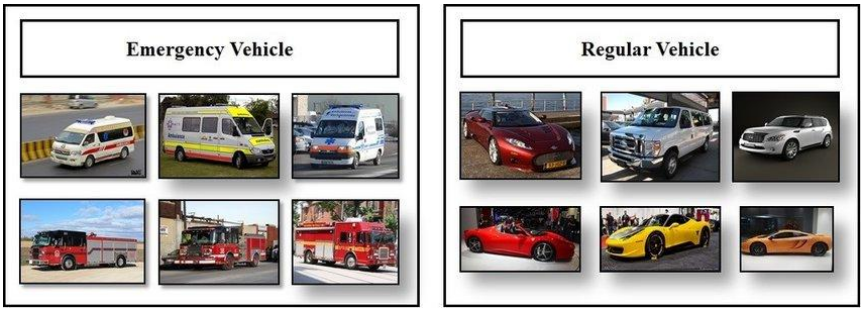


Figure 2.3.4 Vehicle Type

Thirdly, to manage the high processing requirements, simpler models or more compact models should be implemented. Utilize hybrid systems that combine rule-based logic with lightweight AI for efficiency and speed. Employing local computing or distributing out the processing across several systems can help minimize delays while improve the system to work more effectively and be more cost-efficient in large urban areas. Schedule regular offline training with updated traffic patterns to reduce the need for heavy real-time learning loads.

Lastly, it is essential to consider the way human drivers behave in reality. This can be achieved by using real-time data from camera or sensor to understand and adjust to unpredictable driver behaviour. Use historical driving data and machine learning to identify common reaction delays in dense traffic scenarios. By applying these techniques that can learn and adjust over time will assist the system respond more accurately to human drivers' behaviour and it will also help emergency vehicles pass through the traffic congestion more smoothly. Train the system to recognize non-compliant driver behaviour and adjust emergency routing dynamically to avoid delays.

Figure 2.3.5 Observe real-time traffic control

2.4 Review of Technologies

This section reviews the key technologies employed in the development of the Smart Traffic Light System with Multi-Head Attention Deep Q-Learning (MHA-DQL). Each technology is evaluated in terms of its suitability for the simulation-based reinforcement learning environment used in this project.

2.4.1 Hardware Platform

The development and training of this system were carried out entirely on a laptop computer. Since the system is simulation-based, no physical embedded hardware such as Raspberry Pi, Arduino, or external sensors was required. The hardware platform used is the MSI Bravo 15 laptop, equipped with an AMD Ryzen 5 5600H processor, 8 GB DDR4 RAM, an AMD Radeon RX 5500M GPU (4 GB GDDR6), and a 512 GB NVMe SSD.

The AMD Ryzen 5 5600H is a six-core, twelve-thread processor capable of handling the multi-threaded demands of running three simultaneous SUMO simulation instances alongside the PyTorch neural network training loop. The dedicated GPU accelerates matrix operations within the Multi-Head Attention network layers through CUDA-compatible computation. The 512 GB NVMe SSD provides sufficiently fast read and write throughput for frequent model checkpoint saves and JSON training data exports throughout the 600-episode training run.

2.4.2 Firmware / Operating System

The system runs on Microsoft Windows 11 as its operating system. Windows 11 was selected because it provides native support for the SUMO simulation software and its GUI tools, including NETEDIT and the SUMO graphical interface. It also supports the AMD graphics driver stack required to activate GPU-accelerated computation via PyTorch's CUDA backend.

Python 3.10 serves as the runtime environment, managed through a standard pip installation. The SUMO environment variable SUMO_HOME is configured system-wide under Windows 11 to allow the TraCI Python API to locate the SUMO tools directory at runtime. No containerisation or virtual machine environment was used, as the simulation requires direct GPU and display driver access.

2.4.3 Database

This project does not use a traditional relational database management system (RDBMS) such as MySQL or SQLite. Instead, training metrics and episode data are persisted using JSON

(JavaScript Object Notation) file format through Python's built-in json module. At the end of each training session, all per-episode data including total reward, average waiting time, queue lengths by direction, emergency vehicle waiting times for all three agents, and model loss values are serialised and written to a file named training_data.json.

JSON was chosen over a full RDBMS because the data access pattern is sequential and write-once: metrics are appended per episode and read back offline for analysis and plotting. JSON files are human-readable, require no database server process, and can be loaded directly into Python data structures for analysis using the json.load() function. For future extensions involving multi-intersection networks or real-time dashboards, a time-series database such as InfluxDB or a lightweight SQLite database would be more appropriate.

2.4.4 Programming Language

Python 3.10 is the primary programming language used throughout this project. Python was selected for three main reasons. First, the TraCI (Traffic Control Interface) API, which is the only supported interface for real-time programmatic control of SUMO simulations, provides an official and well-maintained Python library. Second, PyTorch, the deep learning framework used to implement the Multi-Head Attention DQN and Vanilla DQN networks, is Python-native and provides automatic differentiation, GPU tensor operations, and a flexible module system. Third, Python's scientific computing ecosystem (NumPy, Matplotlib, collections, random) provides all the utilities needed for state processing, experience replay buffer management, and training progress visualisation.

The entire codebase, including the simulation environment classes (TrafficEnvironment, EnhancedTrafficEnvironment), the neural network definitions (EnhancedTrafficDQN, VanillaTrafficDQN), the agent classes (EnhancedDQNAgent, VanillaDQNAgent), the baseline runner (BaselineEnvironmentRunner), and the training coordinator (EnhancedTrafficTrainer), is implemented in a single Python script to simplify deployment and dependency management.

CHAPTER 3 SYSTEM METHODOLOGY AND APPROACH

3.1 Overview of Methodology

This project uses a simulation-based AI method to solve the ongoing challenges of Heavy traffic congestion in populated areas that are caused by the inefficient signal timing. In particular, the project mainly uses SUMO (Simulation of Urban Mobility) together with Multi-Head Attention Deep Q-Learning (MHA-DQL) algorithm to design and develop a smart traffic signal management system. The system can dynamically prioritize emergency vehicles at junctions. This method includes several advanced technologies and methods, which play a key role in achieving the project goals. SUMO is an open source which is used as a detailed traffic simulator that is utilized to build the base of this project development process and testing setup. It has strong controls and details on traffic system design features which make it the best tool for replicating the real-time traffic conditions within a managed simulation. By using SUMO, the system allows to create various traffic conditions such as roads with multiple lanes or junctions, designing custom traffic situations and inserting various types of vehicles like cars, buses, ambulances, lorries, fire trucks, police vehicles. The simulator allows the system to customize traffic signal logic and the real-world extraction of traffic information using TraCI (Traffic Control Interface) module, which provides Python APIs for smooth interaction between the model and external artificial intelligence programs.

The traffic control rules in this project are powered by a Deep Q-Learning (DQL) algorithm, a reinforcement learning method where program learns best steps by interacting with the system and getting feedback in the form of rewards. The program in this case, is responsible for choosing the most useful timing and sequence of traffic light phases to minimize overall vehicle waiting time and increasing the intersection flow. The data space for the program has various traffic metrics obtained from SUMO, such as the number of waiting vehicles, their positions, and emergency vehicles presence. For the better improvement of the smartness and responsiveness of the system, a Multi-Head Attention (MHA) mechanism is integrated into the DQL architecture. Attention mechanisms have shown great success in natural language processing and sequence learning, and in this context, they allow the agent to focus on different features or components of the traffic state at the same time. For example, one part of the model is to focus on emergency vehicles prioritization, another part is to check vehicle queue length, and another can look at past traffic patterns. This helps the system understand complex traffic situations better and make smarter decisions. In summary, this

methodology shows a useful and forward-thinking way to smart traffic control system. By combining the powerful simulation tools of SUMO with the flexible decision-making strength of Multi-Head Attention Deep Q-Learning, this project scalable to city traffic problems. This way not only ensures technical strength but also matches with the broader goal of creating smarter, safer, and more sustainable using artificial intelligent based traffic control systems

3.2 Reinforcement Learning Problem Definition

The traffic signal control problem of the current project can be defined as a Markov Decision Process (MDP), denoted as (S, A, R, γ). The agent develops an optimum control policy by constantly engaging with the SUMO simulation world via the TraCI interface.

- **State Space (S):** The state is a dual representation: a spatial state $\mathbf{S}_t \in \mathbb{R}^{4\times25}$ representing the current traffic conditions of the four intersection approaches, and a temporal state $\mathbf{T}_t \in \mathbb{R}^{10\times25}$ to keep track of the changes in the traffic conditions. The complete state description is presented in Section 3.4
- **Action Space (A):** The agent chooses a signal phase and a duration of green time at every step of a decision. There are four signal phases ($N_{\text{phases}} = 4$), and eleven different green time durations ($N_{\text{durations}} = 11$) that can be selected between 10 s and 60 s with a 5 s sacrifice. The number of discrete action space is: $|\mathcal{A}| = \mathrm{N}_{\text{phases}} \times \mathrm{N}_{\text{durations}} = 4 \times 11 = 44$.
- **Reward Function (R):** The reward signal R_t is a combination of several functions that maximize traffic throughput, keep vehicles moving, minimize queue length, provide equity among all directions and give priority to emergency vehicles. The entire formulation of the reward is described in Section 3.5.
- **Discount Factor (γ):** $\gamma = 0.99$ is used, so that the agent considers the efficiency of dealing with traffic in the long term nearly equal to the immediate reward, and will make decisions that avoid future congestion as opposed to making decisions that only respond to the present circumstances.

3.3 System Design

This chapter shows the complete system design of a smart traffic light system with multi-head attention deep Q-learning (MHA-DQL). The system builds out the prototype into three agent's comparative evaluation framework of the MHA-DQN agent, a Vanilla DQN ablation agent and

a fixed-time Baseline controller with all being run in parallel as independent SUMO simulation instances and evaluated under identical traffic scenarios.

The following diagrams illustrate the four perspectives of the system: architecture of the system and agent environment integration showing the relationships between the various components of the system, use case diagram is showing the relationships between actors and the activity diagrams showing the operational flow of three agents during operation.

3.3.1 High-Level Architecture Diagram

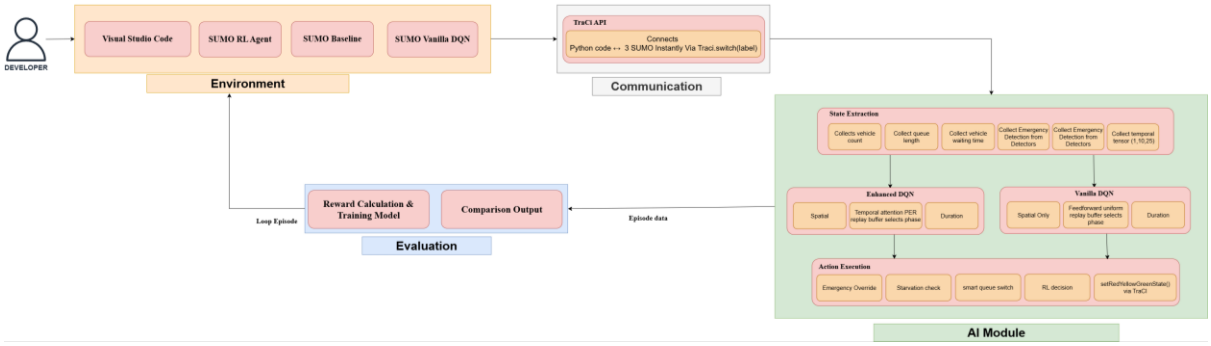


Figure 3.3.1 – High-Level System Architecture

As shown in the diagram above, the architecture of the proposed Smart Traffic Light System is composed of four distinct layers which is **Environment, Communication, AI Module, and Evaluation.** These layers will be used collectively to simulate real time traffic conditions, gather and process real time data about traffic conditions, and intelligently make decisions about which light signal to change, ultimately evaluating the effectiveness of the system.

Three SUMO simulators are included in the Environment Layer, running simultaneously. The first simulator, SUMO RL Agent, is an implementation of the proposed reinforcement learning algorithm, and it forms the training environment. It learns about the optimal traffic signal strategy by interacting with the environment. The second simulator is known as the SUMO Baseline model and employs the traditional fixed-time round-robin approach to traffic lights. In this case, the traffic lights have the same time interval in the green phase. It is implemented to provide a baseline for the other models. The third simulator is referred to as the SUMO Vanilla DQN and uses a standard Deep Q-Network with a replay memory buffer of spatial traffic data.

The Communication Layer serves to link the Python controller program and the SUMO simulations using the TraCI interface. TraCI facilitates real-time communication between Python and SUMO. This makes it possible for the system to access data on traffic parameters like vehicle flow rate, queue length, and wait time from the SUMO simulation, while at the same time making it possible for Python to manipulate the traffic signals in terms of their phases, timings, and priority.

AI Module Layer is tasked with evaluating traffic dynamics and determining the optimal signal action. First, there is an input processing stage where the model extracts state information from inputs like traffic volume, queue length, wait time, presence of emergency vehicles, and temporal data on traffic. These states undergo evaluation by two models: Enhanced DQN and Vanilla DQN. While the Enhanced DQN employs spatial, temporal attention, priority experience replay, and adaptive duration selection, Vanilla DQN is composed of spatial inputs with feedforward architecture and random experience replay.

Following the determination of the best course of action, the next stage involves its execution, which is carried out by Action Execution. This entails actions such as emergency override, starvation avoidance, smart queue switching, reinforcement learning, and changing the state of traffic signals using TraCI.

The Evaluation Layer evaluates the performance of all the models after every episode. Reward scores are obtained depending on traffic efficiency, followed by the comparison of RL Agent, Baseline, and Vanilla DQN. The results are employed in evaluating the performance of the developed smart traffic light system.

3.3.2 Agent Environment Integration

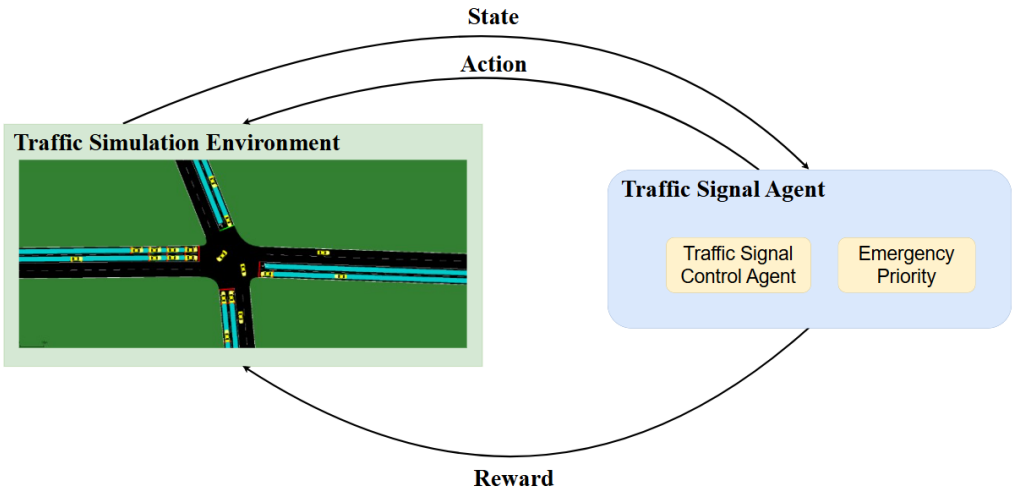


Figure 3.3.2 Reinforcement Learning Model Diagram

The proposed system above has shown the traffic signal control that have been modelled through reinforcement learning system. In this Figure 3.3.2, the **Traffic Simulation Environment** has been represented the simulation road environment using SUMO where vehicles such as regular cars, and emergency vehicles that travel through the intersections. The environment produces the **state** that has contain data such as vehicle speed, vehicle queue length, and the occurrence of the emergency vehicles like ambulances or fire trucks. This state will be delivered to the **Traffic Signal Agent** that consists of two main part which is traffic signal control agent and the emergency vehicle prioritization. Based on the current state the agent will select an **action** for example to adjust the signal phases or to prioritize the emergency vehicles to pass the intersection quicker. The traffic simulation environment then delivers a **reward**, that has been calculated from performance measures such as minimizing the waiting time, reducing the queue length, and successful emergency vehicle's lane clearance. This continuous loop of state, action, and reward allows the system to train and learn the optimal control methods for managing traffic movement efficiently.

3.3.3 Use Case Diagram and Description

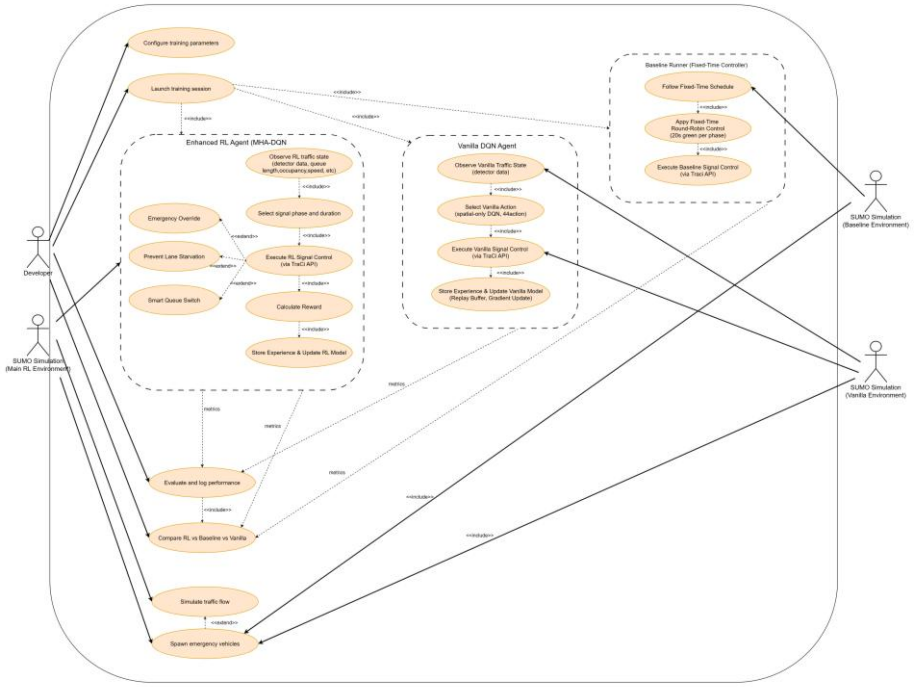


Figure 3.3.3 Use Case Diagram of the Smart Traffic Light System

The diagram above shows the Use Case Diagram of the system. This system has a total of four different types of actors: Developer, SUMO Simulation (Main RL Environment), SUMO Simulation (Baseline Environment), and SUMO Simulation (Vanilla Environment). It is the task of the Developer to configure the hyperparameters for training using the CONFIG dictionary, which includes the number of phases, green phase duration, temporal length, and steps per episode. Post-configuration, the Developer initiates the training using the EnhancedTrafficTrainer class that initializes the simulations and the agents simultaneously.

The first subsystem consists of the Enhanced RL Agent (MHA-DQN). It is implemented through the EnhancedDQNAgent and EnhancedTrafficDQN classes. Firstly, the agent senses the state of traffic and gathers data from detectors on the queue length, occupancy, average speed, and number of vehicles on all four lane approaches. Then, the agent makes a decision about which signal phase and how long to keep it green with the help of a Multi-Head Attention neural network. The actions are performed through the TraCI interface. There are three conditional extensions for the agent's execution: Emergency Override sets up the green phase if there is an emergency vehicle within 150 meters of the intersection, Prevent Lane Starvation skips the agent's action if it is already three steps since the lane was last served, and Smart

Queue Switch switches the green phase to the busier lane if its queue length exceeds double the size of the queue in the agent's action lane.

Secondly, the Vanilla DQN Agent, contained within the VanillaDQNAgent class, acts as a reinforcement learning baseline. In contrast to the Enhanced RL Agent, it does not take into account any temporal information or attention mechanism and relies solely on the current spatial state for decision-making, training via a replay memory where data is uniformly sampled. The evaluation of this agent takes place after every 10th episode in order to compare it with the MHA-DQN.

Thirdly, the Baseline Runner, contained within the BaselineEnvironmentRunner class, acts as a traditional controller. As opposed to the other agents, it implements a round-robin time schedule with 20 seconds of green light for each phase irrespective of traffic. Emergency Vehicle Spawner is in simulations, generating 3 to 5 emergency vehicles during each episode in the simulations. During the evaluation process, data like waiting time, queue length, reward, and emergency vehicle waiting time is gathered from all three simulations. This information is used for the Comparison RL vs Baseline vs Vanilla scenario, wherein learning curves and graphs for comparing the smart traffic light system’s performance are created.

3.3.4 Activity Diagram

3.3.4.1 Activity Diagram of the MHA-DQN Agent Training Process

The diagram below shows the activity diagram of the MHA-DQN agent training process, and demonstrates how the agent observes the dual-stream state, selects an action using an $\varepsilon-$ greedy policy, executes the chosen phase and duration in SUMO, receives a composite reward, stores the experience in the prioritized replay buffer, and updates the Q-network through minibatch learning.

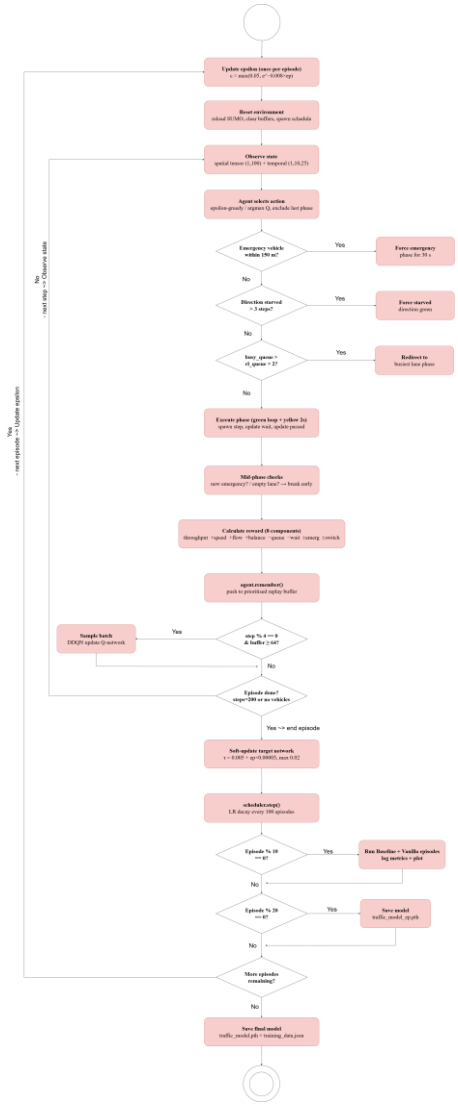


Figure 3.3.4.1: Activity Diagram of the MHA-DQN Agent Training Process

The diagram above illustrates activity diagram of the MHA-DQN agent training procedure, which shows the sequence of events that happen during the training of the Enhanced RL Agent per episode and at each decision point.

At the beginning of every episode, the exploration rate of the agent is updated according to the following formula: $\varepsilon = \max(0.05, e^{-0.008} \times \text{ep})$ This happens in the method **update_epsilon()**. It uses the principle of exponential decay to reduce randomness over time as training continues, so that the agent exploits the acquired knowledge in the form of learned Q values. Then, the environment is reset with SUMO simulation being restarted, the history buffers being emptied, and the spawning of emergency vehicles planned for the episode with help of the class EmergencyVehicleSpawner. Then, the agent observes the current state of traffic that includes the spatial tensor of size (1,100), containing information about the state of

traffic detected, it decides on its next move based on the current observation in an epsilon-greedy fashion, making a decision between a randomly chosen action or a non-previously chosen Q values' argmax value.

Prior to the action being performed, there are three override checks in order that need to be done. First, if there is any presence of emergency vehicles within a radius of 150 metres of the intersection, the selected green phase is overridden for 30 seconds. Second, if there have been at least three sequential steps where one of the lane directions is starved for green time, the action is overridden and the green phase is forced on this direction. Lastly, if the busier lane queue size is greater than double the queue size of the agent's action, the selected phase is overridden. In case no overrides occur, the agent's action is executed.

After selection, the chosen phase will be carried out using the green loop followed by 2 seconds of yellow using the TraCI API. During execution, mid-phase evaluation keeps checking for the presence of an emergency vehicle and whether the lane is empty; in case any of these is found, then the phase will terminate prematurely. Post execution, reward will be evaluated using 8 parameters that include throughput, speed, flow, balance, penalty for queues, penalty for waiting, emergency rewards, and penalties for switching phases. The experience is stored in the prioritised replay buffer via agent.remember().

Once every 4 timesteps, the buffer is checked to see if it has collected more than 64 experiences, then a batch is sampled from the buffer, and the Q-network is trained via double DQN with prioritized experience replay. The game will end once the number of timesteps hits 200 or there are no more vehicles left in the environment. Upon completion of each episode, the target network will be updated softly with a value of τ set to $\min(0.005 + \text{ep} * 0.00005, 0.02)$. The learning rate schedule will decay the learning rate once every 100 episodes. Every 10 episodes, the Baseline and Vanilla DQN environments will be tested, and the data will be logged and plotted. Every 20 episodes, the model will be saved to traffic_model_ep.pth.

3.3.4.2 Activity Diagram of the Vanilla Training Process

The activity diagram of the Vanilla DQN training process shows how the agent only sees the current state of space, without seeing the past, and then chooses an action using an ε-greedy policy, executes the phase and duration in another instance of SUMO, receives a simple reward, and updates the Q-network with random experience replay without prioritization.

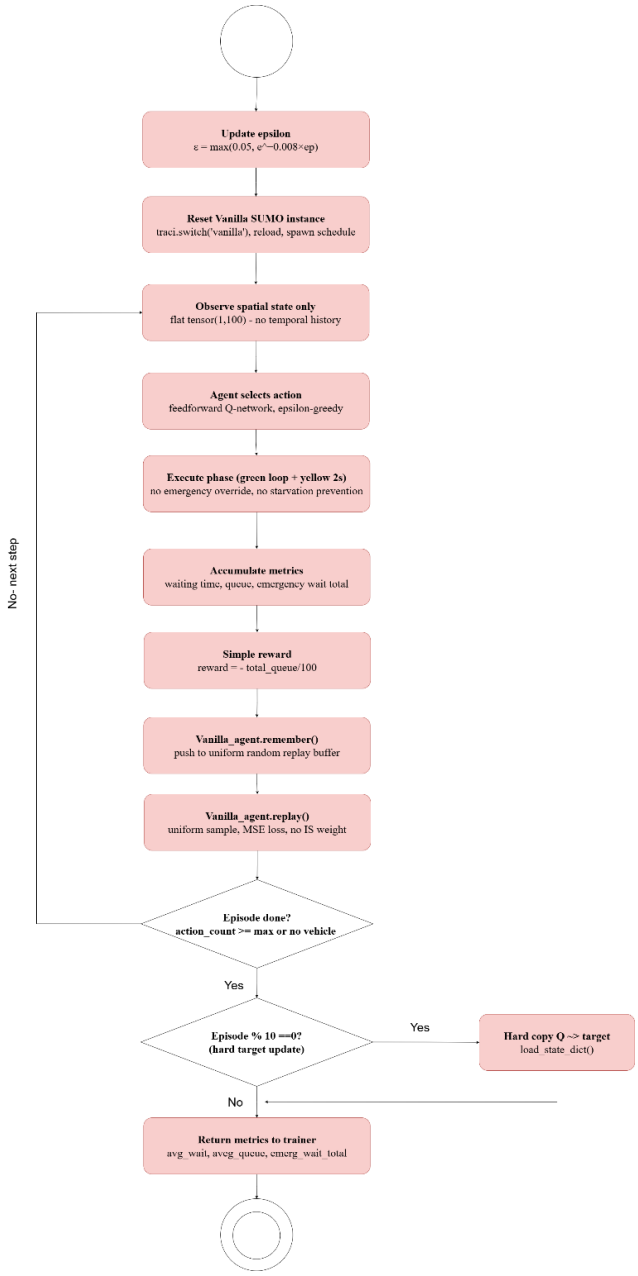


Figure 3.3.4.2 Activity Diagram of the Vanilla DQN Agent Training Process

The activity diagram shown above represents the training process of the Vanilla DQN agent, providing a description of the workflow of the VanillaDQNAgent object with respect to the environment provided by VanillaEnvironmentRunner class.

This agent has been programmed to update its value of ε based on the same equation which is $\varepsilon = \max(0.05, e^{-0.008} \times ep)$. After every episode is run, a new version of the Vanilla SUMO model is created through traci.switch ('vanilla').

Contrary to the MHA-DQN agent, which perceives both the spatial state and temporal history along with the attention component, the Vanilla DQN agent has access only to the spatial state in form of a plain tensor of size (1,100). From the obtained observation, the agent decides its next move via an epsilon-greedy strategy applied to a conventional feedforward Q-network. The decided phase is activated using a green signal, followed by a yellow transition lasting for 2 seconds via the TraCI API.

After finishing each episode, the metrics include queue length, wait time, and wait time for emergency vehicles. These are obtained via _accumulate_metrics(). The computation of the simple reward is performed according to the formula: $\text{reward} = -\text{total\_queue}/100$. The experience is then stored in SimpleReplayBuffer, where uniform sampling is used. Lastly, the network is trained via vanilla_agent.replay(). This is based on MSE loss with no weighting, which contrasts with the weighted loss for prioritized experience replay in MHA-DQN.

At the end of an episode, either the action limit has been reached or there are no more cars left. After every ten episodes, the target network will be updated by hard copy using the load_state_dict(). Lastly, the performance measurements such as average wait time, queue length, and emergency car wait time are provided back to the trainer.

3.3.4.3 Activity Diagram of the Baseline Training Process

The diagram shows the activity diagram of the Baseline training process and how the fixed-time signal controller cycles through all four phases in a fixed time sequence of 20 seconds per phase with no learning or adaptation to the actual real-time traffic conditions.

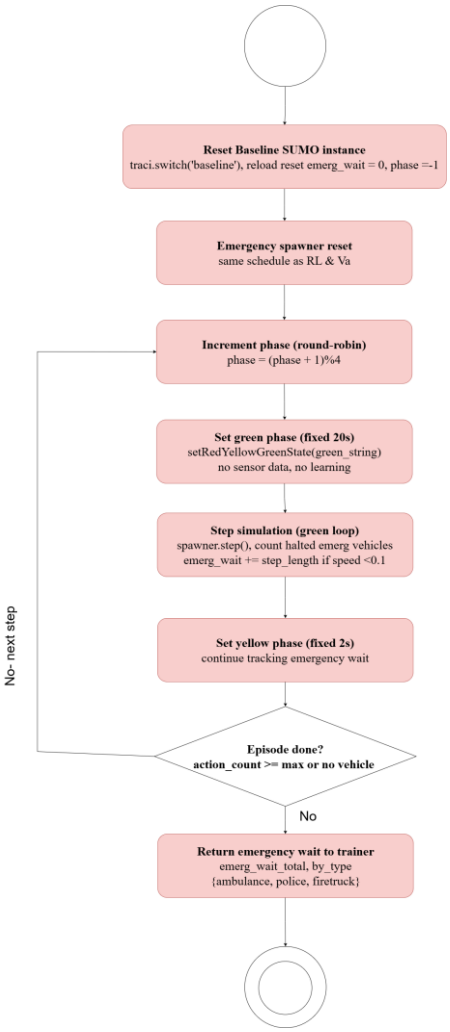


Figure 3.3.4.3 Activity Diagram of the Baseline Fixed-Time Controller

The diagram above shows the activity diagram for the Baseline Fixed-Time Controller which has been designed in the class BaselineEnvironmentRunner.

At the beginning of each episode, the simulation is loaded from the Baseline SUMO environment by using traci.switch('baseline') command along with initializing emergency waiting time to zero and phase counter to -1. Finally, the emergency spawner is initialized to the same schedule used for the RL and Vanilla simulations.

Now, the controller begins a straightforward round-robin process. In each cycle, it increases the value of the phase using phase = (phase + 1) % 4, sets the value of the green phase for a

total of 20 seconds directly with **setRedYellowGreenState()** without requiring any input from sensors or any kind of learning process. Within the green phase, simulation steps continue, the emergency spawner gets activated with every step, and waiting time of the emergency vehicles is counted if their speed becomes lower than 0.1m/s. The yellow phase lasts for 2 seconds.

The episode terminates either upon reaching the limit of actions or depletion of vehicles. The total waiting time for emergencies in each episode, categorized by the type of vehicle — ambulance, police vehicle, and fire truck, is then delivered to the trainer as output.

3.4 State Space Definition

The agent uses two complementary representations of the environment, namely a spatial state that holds up-to-date traffic information on all four intersection approaches and a temporal state that holds the previous 10 observations of the environment to understand traffic evolution trends. This duality is achieved in this via get_spatial_state() and the temporal history deque.

3.4.1 Spatial State Vector

In each direction of the approach d of the set of directions $d \in$ {West,East,North,South}, four real-time measurements are provided by the SUMO lane area detectors. Such are determined in the get-detector-data method.

$$\mathbf{f}_d = [q_d, o_d, v_d, c_d] \in \mathbb{R}^4$$

where:

- q_d = queue length— length of vehicles with a speed that is less than 0.1 m/s
- o_d = occupancy — proportion of time that the detector is occupied (range 0 to 1)
- v_d = mean speed — average speed of vehicles in m/s
- c_d = vehicle count — total number of vehicles currently on approach d

These four characteristics give a detailed overview of the traffic conditions in every approach road. The queue length q_d is a direct measure of the severity of congestion, as it is the number of vehicles that have come to a complete stop. The occupancy o_d is used to determine the density of the vehicles packed together and thus to identify a near gridlock situation when the vehicles are still crawling. Mean speed v_d in metres per second can help the agent to differentiate free-flowing traffic and congested conditions. Vehicle countc_d adds to the queue length by incorporating moving vehicles that are yet to be in the queue. The combination of these four values results in the foundation of the situational awareness of the agent, which

allows it to distinguish between a lightly loaded green phase and a seriously overloaded red phase.

3.4.2 Global intersection features

The agent also monitors the global intersection conditions, in addition to per-direction:

$$\mathbf{g}_t = [Q_t, C_t, p_t, \tau_t, p_{t-1}] \in \mathbb{R}^5$$

where:

- $Q_t = \sum_d q_d$ = sum of queue length in all directions
- $C_t = \sum_d c_d$ = number of vehicles in the network.
- $p_t \in \{0,1,2,3\}$ = current active phase index
- $\tau_t = \frac{t_{\text{phase}}}{T_{\max}}$ = elapsed time in current phase normalized by maximum green duration
- p_{t-1} = transition phase (for smooth transitions)

Global characteristics provide the agent with a bird-like perspective of intersection-wide state as opposed to mere per-approach shots. The length of the total queue Q_t will make the agent aware of the fact that the intersection is getting congested, and it may require more aggressive actions by clearing the queue. The total number of vehicles C_t gives the same information but also takes into account vehicles in motion but has not stopped yet. The phase index p_t and normalized elapsed time τ_t inform the agent of how much time the current green has been on. This is paramount in determining to either extend the green or shift to a new phase. Previous stage p_{t-1} enables the agent to be aware of the transitions and prevent oscillatory behavior.

3.4.3 Historical phase distribution

To promote even service in all directions, the agent monitors the frequency of usage of each phase in the recent past. This is done within the get_spatial_state() method in which maintains a green history deque and calculates normalized frequencies.

$$\mathbf{h}_{\text{green}} = \left[\frac{\text{count}_0}{H}, \frac{\text{count}_1}{H}, \frac{\text{count}_2}{H}, \frac{\text{count}_3}{H}\right] \in \mathbb{R}^4$$

where count_i is the count of phase i that has been hit in the past $H = 10$ steps, and $\mathbf{h}_{\text{green}}$ is the sum of the counts.

The distribution of this phase in history does not allow the agent to work with the same direction repeatedly, neglecting the others. This is what is referred to as lane starvation. When the agent is biased to a specific stage due to high immediate rewards, the normalized frequency of that stage will tend to be close to 1.0 and others close to 0. This imbalance can be then punished by the balance component of the reward function on the agent. The sliding window of length $H = 10$ can be used to make sure that only the recent history is important that the ancient selections of phases are lost, and the agent can adjust to the changing traffic patterns without being punished because of making outdated choices.

3.4.4 Full spatial state tensor

The spatial state is structured as: Since the model processes all four directions in parallel using Multi-Head Attention, the spatial state is structured as:

$$\mathbf{S}_t = [\mathbf{s}_{\text{West}}, \mathbf{s}_{\text{East}}, \mathbf{s}_{\text{North}}, \mathbf{s}_{\text{South}}]^\top \in \mathbb{R}^{4\times 25}$$

When being flattened to feed into some network layers (in particular the Vanilla DQN baseline that does not have attention), the tensor is reshaped to:

$$\mathbf{s}_t^{\text{flat}} = \text{flatten}(\mathbf{S}_t) \in \mathbb{R}^{100}$$

Multi-Head Attention mechanism takes the 4×25 tensor form as its native form. The rows are matching the intersection approaches, and the columns are the features. Relationships between rows are then computed by the attention mechanism. As an illustration, the length of the queue on West (row 0, column 0) versus the length of the queue on East (row 1, column 0). This relational argument cannot be done in the case of a flat vector. The flattened 100-dimensional vector is instead sent to the Vanilla DQN baseline, with no attention, which means that the system treats this difference in a clean manner, For example the MHA-DQN is run in the tensor form whereas the comparison agents are run in the flattened form, thus allowing a fair comparison between them.

3.4.5 Temporal State Tensor

In order to capture the time dynamics of traffic, the agent uses a deque of the previous $H = 10$ spatial states. This is applied using self.temporal history = deque(maxlen=10).

$$\mathbf{T}_t = [\mathbf{S}_{t-H+1}, \mathbf{S}_{t-H+2}, \ldots, \mathbf{S}_t] \in \mathbb{R}^{H \times 4 \times 25}$$

where $H = 10$ is the temporal window length.

The agent can only view the current snapshot, without temporal history, which it would have no clue whether a long queue just appeared within the last second or has been increasing the past minute. The agent has a 10-step history, so the agent can learn trends: say, the temporal attention mechanism can learn that West queue length has been rising over the past 4 steps is a pattern that predicts future congestion. The $H = 10$ tradeoff has been chosen because longer histories would have a higher cost in computational cost and information value, but the returns to length will be diminishing, as traffic patterns further ahead than the last 10 decision steps (around 100-600 seconds of real time) are less relevant to the near decision.

3.5 Reward Function Design

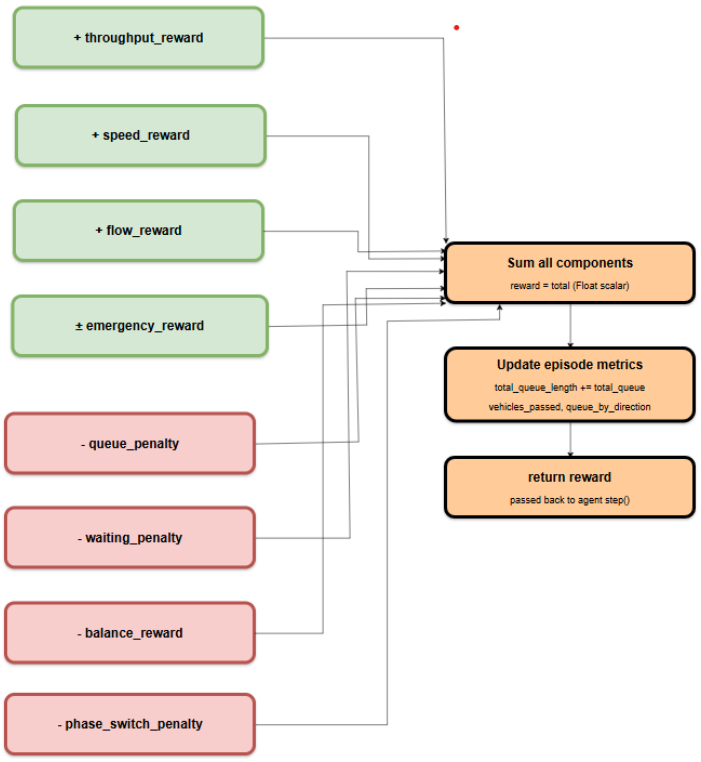


Figure 3.5 Reward Function Design Diagram

The most important element that guides the learning of the agent is the reward function. It is applied within the calculate_action reward() method that occupies about 100 lines of code and considers eight different elements. An effective reward should strike a balance between various, even conflicting, goals: clearing the vehicles fast, reducing the waiting time, being fair along all directions, and giving the emergency vehicles a first priority.

The reward function

$$R_t = R_{\text{throughput}} + R_{\text{speed}} + R_{\text{flow}} + R_{\text{balance}} - P_{\text{queue}} - P_{\text{waiting}} + R_{\text{emergency}} - P_{\text{switch}}$$

The last reward is the number of good incentives over penalties. Positive components promote desirable behaviors: throughput rewards vehicles cleared, speed rewards vehicles maintained, flow rewards vehicles velocity at intersection, and emergency rewards vehicles of importance. Punishments are used to deter unwanted behaviors: build up of queues, long waiting time, unequal distribution of queues and wasteful phase switching. All the components are meticulously balanced such that none of the components overtakes a typical episode; throughput and emergency rewards usually add +10 to +30, whereas penalties deduct -5 to -15. This balance was resolved by experimentation with the scaling factors (0.3, 0.5, 0.2, etc.) that maintain the range of the total reward in a reasonable range.

3.5.1 Throughput Reward

This compensates the agent when he or she manages to pass through the intersection with vehicles. It calculates this by the number of vehicles on each detector at the start and end of the phase.

$$R_{\text{throughput}} = \sum_{d \in D} \max\left(0, \frac{c_d^{\text{prev}} - c_d^{\text{curr}}}{5.0}\right) \times 1.0$$

where $D = \{\text{West,East,North,South}\}$, c_d^{prev} is the number of vehicles at the start of the phase and c_d^{curr} is the number of vehicles at the end of the phase.

Assuming that there were 10 vehicles on the West approach during the green phase and that there are still 2 vehicles thereafter, then there will be 8 vehicles cleared. This is normalized

by dividing by 5.0 to +1.6 reward. The 5-division factor was selected as the typical throughput per phase is between 0 - 25 vehicles; a division factor of 5 would ensure that reward contribution is within 0 to +5. The max(0) definition will make sure that the agent does not receive rewards on vehicles arriving at the phase (i.e. vehicles entering the crossroad, not passing through it). The directional reward structure will motivate the agent to focus on the steps that advance traffic and not just holding a green on a deserted lane.

3.5.2 Speed Reward

This will promote the movement of traffic at moderate speeds instead of crawling. Speed reward is based on the average speed of each detector but limited to the speed limit.

$$R_{\text{speed}} = \sum_{d \in D} \min\left(\frac{\bar{v}_d}{13.89}, 1.0\right) \times 0.3$$

where $\bar{v}_d$ is the mean speed on approach d in m/s, and 13.89 m/s = 50 km/h (the speed limit).

Assuming cars are travelling with the speed limit (13.89 m/s or 50 km/h), the ratio is 1.0, which has the maximum speed reward of 0.3 per direction. When vehicles are moving at 25 km/h (6.94 m/s), the ratio will be 0.5 (reward 0.15 per direction). The min (1.0) limits the ratio to ensure that going faster than the speed limit does not help: This avoids the promotion of unsafe driving. The 0.3 multiplier was empirically selected because it was desired to maintain the magnitude of the speed reward similar to other reward elements. The possible speed reward, in all four directions, has a maximum of +1.2 per step, significant but not dominant.

3.5.3 Flow Reward

This offers further incentive to total intersection throughput, augmenting the directional speed reward, by averaging the speed of all approaches.

$$R_{\text{flow}} = \left(\frac{1}{|D|} \sum_{d \in D} \bar{v}_d\right) \times 0.2$$

Where the speed reward is already an incentive to high speeds on any individual approach, the flow reward is a worldwide indicator of a reward which is given to the average speed of all four directions. The flow reward will be high when all the four approaches are high in speed and when some of the approaches are congested it will be low. This component is provided by the multiplier 0.2 a weight of about 2/3 that of the per-direction speed reward (sums up to 1.2).

This balance makes sure that the agent is concerned about the performance of each lane and the fluidity of the intersection, so it does not attempt to maximize a direction by sacrificing the other.

3.5.4 Balance Reward (Fairness)

This punishes uneven distribution of queues in directions to avoid starvation.

$$R_{\text{balance}} = -\sigma(\{q_d\}_{d \in D}) \times 0.5$$

where $\sigma(\,)$ is the standard deviation of queue lengths across West, East, North, South. If one direction has 50 queued vehicles and others have 0, standard deviation is large to negative reward encourages balancing.

3.5.5 Queue Penalty

This punishes the agent by letting long lines build up. The total queue length is the amount of queue length of all the four approaches.

$$P_{\text{queue}} = \left(\frac{Q_t}{100.0}\right) \times 0.5$$

where $Q_t = \sum_d q_d$ is the total queue length across all directions.

Assuming the 200 vehicles are lined up on all approaches e.g. 50 per direction, then the penalty would be (200/100) x 0.5 = 1.0. The penalty is 0.05 in case there are only 10 vehicles in line. This linear scaling implies that, as the queue length is decreased by half to 100, the penalty also decreases by 0.5 which is quite a significant signal. The normalization factor of 100 was made due to the fact that the maximum length of queues in the simulation was found to be between 0 and 300 vehicles in all the directions, which held the penalty at a range of 0 to 1.5.

3.5.6 Waiting Penalty

This punishes cars that have stalled fully (speed less than 0.1 m/s) especially non-emergency cars. In the emergency reward component, emergency vehicles are treated differently.

$$P_{\text{waiting}} = \left(\frac{H_t}{10.0}\right) \times 0.2$$

where H_t is the number of halted (speed < 0.1 m/s) non-emergency vehicles.

There are 50 non-emerging vehicles that are entirely stopped; penalty = (50/10) 0.2 = 1.0. Penalty = 0.1 in case 5 are stopped. This is an addition to the queue penalty, which specifically targets totally stationary vehicles and not all vehicles in the queue (which includes slowly moving vehicles). The normalization of the typical halted vehicle counts usually between 0 and 50 per step is done by the divisor 10. This component is weighed appropriately considering other penalties due to its multiplier of 0.2. The fact that emergency vehicles are not counted in this calculation but are dealt with by the emergency reward instead should be noted.

3.5.7 Emergency Reward

This is the most significant element to the objective of the project that is the prioritisation of emergency vehicles. This adopts conditional rewards and punishments depending on the speed of the emergency vehicle, distance to the intersection, and the presence or absence of the appropriate phase.

$$R_{\text{emergency}} = \begin{cases} +5.0 & \text{if emergency speed } > 8 \text{ m/s } (\approx 29 \text{ km/h}) \\ +3.0 & \text{if emergency passed junction and speed } > 12 \text{ m/s } (\approx 43 \text{ km/h}) \\ -\min\left(\frac{w_{\text{emerg}}}{5.0}, 8.0\right) & \text{if emergency speed } < 0.5 \text{ m/s (halted)} \\ -10.0 & \text{if emergency within 50m and wrong phase active} \end{cases}$$

The reward is clipped to the range $[-15, +15]$ to maintain training stability.

This reward framework will give a powerful incentive to the agent to clear ways of emergency vehicles. The positive rewards (+5, +3) are large and promote the agent to continue the emergency vehicles to move fast. The fine of stopped emergency vehicles is proportional to the waiting time, with the highest possible fine -8, which is quite considerable but not disastrous. The largest penalty is -10 when the emergency vehicle is within 50 meters of the intersection, yet the current phase is not its direction which trains the agent to pre-emptively reverse the phase even before the emergency reaches the intersection. The regularization to

[15, +15] helps to avoid the destabilizing effect of extreme reward values on learning. The experiments have demonstrated this multi-tiered strategy with heavier penalties given to erroneous phases than to simple stopping to be very successful, cutting emergency waiting time down to just a few seconds.

3.5.8 Phase Switch Penalty

This punishes the agent when holding a green phase when he/she has no vehicles to serve. The should_switch flag causes this, and it goes to true when a lane has been cleared by the green phase.

$$P_{\text{switch}} = \begin{cases} 3.0 & \text{if lane is empty but agent kept same phase} \\ 0 & \text{otherwise} \end{cases}$$

This penalty will help the agent avoid wasting green time on empty lanes. When the number of vehicles in a lane is zero there is no benefit in holding the green phase which will only delay other directions which may be congested. The punishment 3.0 is not excessive to deter this behaviour but not excessive to the extent that the agent is overly keen to change phases too soon (this may lead to oscillation). This penalty is not applied when the agent is in the opportunity to switch (i.e. the minimum green duration is passed) but anyway did not switch even when the lane was empty.

3.5.9 Total Reward

When all the components are summed up with an appropriate scaling, the reward signal ultimately received by the agent at the end of every phase execution is achieved.

$$R_t = R_{\text{throughput}} + R_{\text{speed}} + R_{\text{flow}} + R_{\text{balance}} - P_{\text{queue}} - P_{\text{waiting}} + \text{clip}(R_{\text{emergency}}, -15, 15) - P_{\text{switch}}$$

The reward is usually between -20 and +30. Negative rewards are used at the time of early training where the agent is not performing well, and long queues, numerous stopped vehicles, and red lights on emergency vehicles. The trend in rewards as learning advances is that, as the agent learns how to efficiently clear queues and maintain sensible speeds, to balance service in both directions and to preemptively respond to emergency vehicles, rewards increase towards positive values. The emergency reward is clipped to [-15, +15] which makes sure that

emergency events do not drive reward magnitudes that overwhelm all the other components and thus make the agent concentrate on the emergencies at the cost of the normal traffic.

3.6 Multi-Head Attention Deep Q-Network Architecture

The proposed MHA-DQN builds on the traditional DQN with three major innovations in the EnhancedTrafficDQN class that is Multi-Head Attention to extract spatial and temporal features, Positional Encoding to maintain sequence order in the temporal data, and DQN with Prioritized Experience Replay that helps reduce the overestimation bias and concentrate learning on significant experiences.

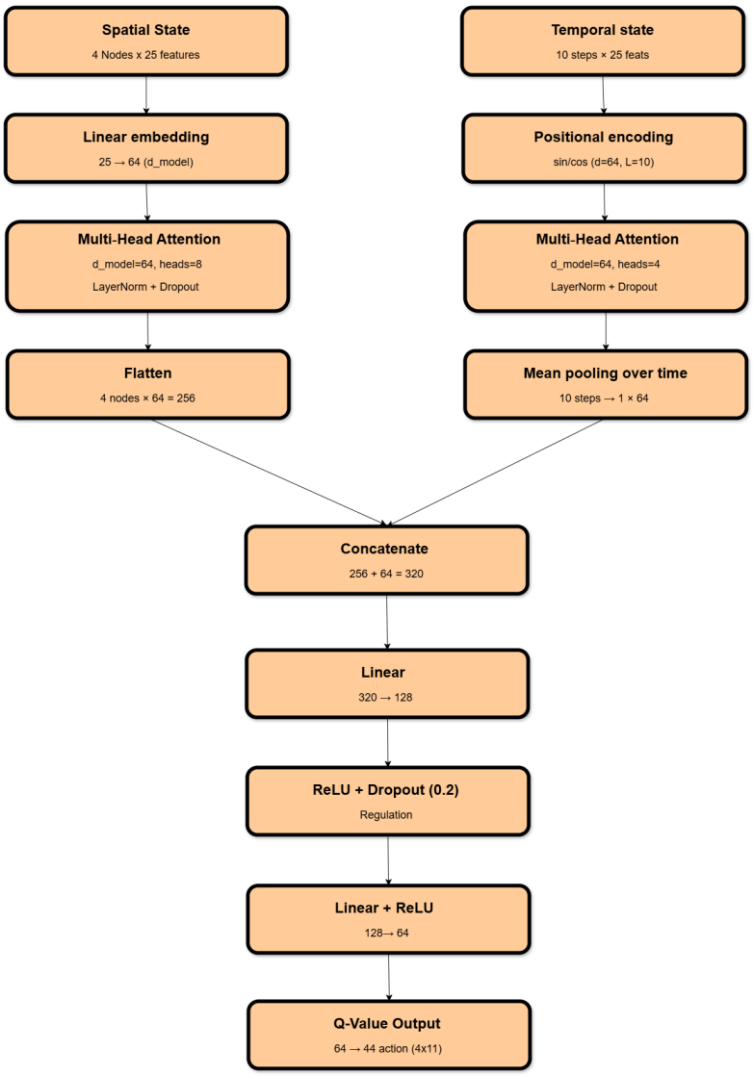


Figure 3.6 Multi-Head Attention Deep Q-Network Architecture

The diagram above represents the Multi-Head Attention Deep Double Q-Network (MHA-DQN). The spatial branch takes the current 4 node traffic state (25 features per node), and runs

it through embedding and 8 head self-attention, flattened down to 256 dimensions. The temporal branch combines the past 10 spatial states, positional encoding and time pooling, to a 64-dim vector. Both branches are concatenated (320 dim) and fed into a two-layer MLP (128 to 64) with dropout and eventually gives 44 Q-values of all pairs of phase-duration actions.

3.6.1 Spatial Attention Encoder

The spatial encoder compares traffic conditions in the four intersection approaches in parallel, enabling the model to compare the congestion levels. This is applied in the spatial embedding and spatial attention layers.

$$\mathbf{X}_{\text{spatial}} = \text{Linear}(\mathbf{S}_t) \in \mathbb{R}^{4 \times d_{\text{model}}}$$

where $d_{\text{model}} = 64$ (embedding dimension) and Linear: $\mathbb{R}^{25} \rightarrow \mathbb{R}^{64}$.

All the four directions (graph nodes) are projected separately to a 64-dimensional embedding space using all its 25 features. This extension provides the model with greater expressive capabilities to model complex traffic patterns. The linear projection is trained; it is initially arbitrary but over time it learns how to transform the raw detector data (queue lengths, speeds, occupancies) into a more useful feature space. The 64-dimensional selection is a trade-off between expressiveness and computational cost of larger dimensionality, which is slower and easier to overfit, and smaller dimensionality, which may not be expressive at all.

3.6.2 Scaled Dot-Product Attention

$$\text{Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}}\right)\mathbf{V}$$

where $\mathbf{Q}$ (queries), $\mathbf{K}$ (keys), and $\mathbf{V}$ (values) are all derived from $\mathbf{X}_{\text{spatial}}$, and $d_k = d_{\text{model}}/h = 8$ is the dimension per head.

This is the basic attention equation. Intuitively, every direction like west would result in a query vector that would pose a question like, "what other directions should i be attentive to? Both directions also generate key vectors which respond to, what information do I have? and value vectors which include the actual information to pass. The dot product of a query and all the keys is a measure of similarity; SoftMax transforms the similarities into attention weights that add up to 1. The result is a weighted sum of the values, i.e. the representation of each

direction is changed by information provided by the rest of the directions. The scaling factor $\sqrt{d_k}$ ensures that dot products in high dimensions are not too large, causing SoftMax to enter regions of very small gradients. This process enables the model to concentrate on the most pertinent directions at every step.

$$\text{MultiHead}(\mathbf{X}) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)\mathbf{W}^O$$
$$\text{head}_i = \text{Attention}(\mathbf{X}\mathbf{W}_i^Q, \mathbf{X}\mathbf{W}_i^K, \mathbf{X}\mathbf{W}_i^V)$$

where:

- $h = 8$ heads (implemented as num_heads=8)
- $\mathbf{W}_i^Q, \mathbf{W}_i^K, \mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_k}$ are per-head projection matrices that can be learnt
- $\mathbf{W}^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}}$ is the output projection

Single-head attention makes the model learn only a single way of relating directions. Multi-head attention enables the model to acquire numerous different relationship patterns at once. Within the traffic application, a head may be trained to compare queue lengths by direction; another may be trained to compare vehicle velocities; a third may be trained to concentrate on occupancy rates; a fourth may be trained to monitor emergency vehicle closeness and so forth. The heads contain their own learned projection matrices hence they are able to pay attention to various features. With all heads computed separately, their results are concatenated and projected to 64 dimensions using $\mathbf{W}^O$. This provides the model with a multi-faceted, deep comprehension of the traffic conditions. The 8 head selection method is common in transformer architectures and has been demonstrated to perform well on sequence and graph attention tasks.

3.6.3 Temporal Attention Encoder

The temporal encoder works with the past $H = 10$ traffic states in order to capture changing trends. This is done in the temporal_embedding, pos and temporal attention layers.

$$\mathbf{X}_{\text{temporal}} = \text{Linear}(\mathbf{T}_t) \in \mathbb{R}^{10 \times 64}$$

where $\mathbf{T}_t$ is the temporal state tensor of shape $10 \times 4 \times 25$ flattened to 10×100 before projection.

The temporal state tensor is a 10×4×25 tensor, consisting of the previous 10 spatial states. To process it with the same attention mechanism, the spatial dimension is first flattened to 100 features per timestep and then projected to 64 dimensions. This results in a 10×64 matrix with each row representing a historical timestep and each column a learned feature. This is an independent per time-step projection whereby the same linear layer is applied to all 10 timesteps sharing weights across time.

Temporal attention block

$$\mathbf{X}'_{\text{temporal}} = \text{LayerNorm}\left(\mathbf{X}_{\text{temporal}} + \text{Dropout}\left(\text{MultiHead}(\mathbf{X}_{\text{temporal}} + \mathbf{PE})\right)\right)$$

where $\mathbf{PE}$ is the positional encoding matrix of shape 10×64.

The model has positional encoding added to the temporal embeddings prior to attention, providing the model with time order awareness. Multi-head attention is then applied to the 10 timesteps, which enables the model to compare various points in history and draw patterns such as queue length increasing in the past 3 steps. The residual connection, dropout, and layer normalisation used in the spatial block are also used, which guarantees stable training and gradient flow.

3.6.4 Q-value output layer

$$\mathbf{Q}(\mathbf{S}_t, \mathbf{T}_t; \theta) = \mathbf{W}_3\mathbf{h}_3 + \mathbf{b}_3, \mathbf{W}_3 \in \mathbb{R}^{44\times 64}$$

where θ represents all trainable parameters of the network.

The last layer produces 44 Q-values, which are the action (4 phases) of the 11 durations. This layer does not have any activation functionality since Q-values may be any real number; positive Q-values denote actions that are estimated to result in future rewards and negative Q-values denote those that are likely to be expensive. The action that has the largest Q-value is the choice that the agent recommends, except in the case that exploration (epsilon-greedy) picks a random action instead. The training of all parameters θ is done with the Adam optimizer to minimize the loss on TD-error.

3.6.5 Reward Function Summary Table

Component	Formula	Weight	Purpose
Throughput Reward	vehicles cleared / 5.0	1.0	Reward agent for moving vehicles through intersection
Speed Reward	min(v/13.89, 1.0)	0.3	Reward maintaining traffic flow at speed limit
Flow Reward	avg speed across all directions	0.2	Reward overall intersection fluidity
Balance Reward	−σ(queue lengths)	0.5	Penalise uneven queue distribution across directions
Queue Penalty	total queue / 100.0	0.5	Penalise long queues forming at intersection
Waiting Penalty	halted vehicles / 10.0	0.2	Penalise stopped non-emergency vehicles
Emergency Reward	conditional ±5.0/+3.0/−8.0/−10.0	clipped ±15	Prioritise emergency vehicle clearance
Phase Switch Penalty	−3.0 if lane empty	3.0	Penalise wasting green time on empty lanes

Table 3.6.5 Reward Function Component Summary

3.7 DQN with Prioritized Experience Replay

3.7.1 Double DQN Target

Standard DQN is affected by overestimation bias as the same network is used to sample and evaluate actions. These functions are decoupled by DQN. This is the replay() method of EnhancedDQNAgent implemented.

$$y_i = r_i + \gamma(1 - d_i) \cdot Q_{\text{target}}\left(s_{i+1}, \arg\max_{a'} Q_{\text{online}}(s_{i+1}, a')\right)$$

where:

- $\gamma = 0.99$ (discount factor — future rewards matter almost as much as immediate rewards)
- $d_i \in \{0,1\}$ indicates episode termination

- Q_{online} (primary network) selects the optimal action
- Q_{target} (target network) assesses that action

The DQN target standard is $\max_{a'} Q_{\text{online}}(s_{i+1}, a')$ both in selection and evaluation, a form that always overestimates Q-values due to the biased nature of the max operator. DQN eliminates this bias by applying the online network to choose the action as the action that is most optimal based on the existing knowledge and the target network to assess the action. This decoupling minimizes overestimation and keeps computational complexity constant. Slowly updating the target network with soft updates provides the target network with a stable learning target. The discount factor of 0.99 implies that the present rewards are emphasized nearly equally with the future rewards and stimulates long-term planning as opposed to shortsightedness.

3.7.2 Prioritized Experience Replay

Standard replay buffer samples randomly, whereas there are more significant experiences to learn (such as arriving at an emergency vehicle). Gives more priority to important experiences in replay. This is done in the PrioritizedReplayBuffer class.

$$p_i = | \delta_i | + \epsilon$$

where:

- $\delta_i = y_i - Q(s_i, a_i; \theta)$ is the Temporal Difference (TD) error
- $\epsilon = 10^{-5}$ (minimal constant that does not make the priority zero)

The TD error indicates the degree of surprise of the agent at the result of a large TD error implies that what the agent predicted was nowhere near the reality, indicating that this experience has a lot of information that is not yet learned by the agent. Replay is prioritized on experiences with large TD errors. The small constant ϵ makes sure that even the experiences with the TD error of 0 correspond to the non-zero priority, thus are still occasionally sampled. Its priorities are updated with every training step as a new TD error is calculated and the update priorities method puts its priorities into the priority array of the buffer again

3.7.3 Action selection policy

$$a_t = \begin{cases} \arg\max_a Q(s_t, a; \theta) & \text{with probability } 1 - \epsilon_t \\ \text{random action from valid set} & \text{with probability } \epsilon_t \end{cases}$$

The valid set excludes the current phase to enforce phase rotation.

The valid set is one which ostracizes the current phase to impose phase rotation. In the exploitation (probability 1-ε), the agent chooses the action that has the greatest Q-value based on the online network that exists at that moment. In the exploration (probability ε) it picks up a random action among the valid actions. The valid set does not include the current stage, so the agent cannot remain in the same stage forever, a failure mode of DQN traffic controllers. This masking promotes a wide range of experiences in the replay buffer by compelling the agent to at least explore other phases and avoids the agent being trapped in a phase in which it might learn policies. This method is applied in the act() method with a mask of actions which gives the current action Q-values equal to negative infinity.

3.8 Emergency Vehicle Preemption

This hard coded override is the most important safety aspect as it makes sure that emergency vehicles are never held up by an intelligent policy which may prioritize other goals. This is done in the implementation of _get_emergency phase and step().

3.8.1: Emergency detection condition

$$\text{Emergency Active} = \exists v \in V_{\text{emerg}}:\text{dist}(v,\text{junction}) \leq D_{\text{preempt}}$$

where:

- V_{emerg} = {ambulance, police, firetruck}
- $D_{\text{preempt}} = 150$ metres

Each step of the simulation, this repeats the process of all vehicles and compares their type to a list of emergency types. In case of any emergency vehicle in the crossroad, the distance between it and the junction is calculated based on the distance to the next traffic light (acquired through traci.vehicle).getNextTLS()). When such a distance is less than or equal to 150 meters, the pre-emption condition is activated. This distance of pre-emption of 150 meters was selected

as it gives a 5-10 second notice during normal approach speeds (15-30 m/s) which is sufficient time to change phases before the vehicle reaches the intersection.

3.8.2 Emergency phase mapping

$$\text{phase}_{\text{emerg}}(d) = \begin{cases} 3 & \text{if vehicle approaching from West} \\ 1 & \text{if vehicle approaching from East} \\ 2 & \text{if vehicle approaching from North} \\ 0 & \text{if vehicle approaching from South} \end{cases}$$

This mapping converts the direction that an emergency vehicle is coming towards to the phase in the traffic light that will grant it a green light. Phases 3 and 4 are assigned to West-bound vehicles and the West-East phase respectively. Phase 1 (East-West) is applied to the east-bound vehicles. Phase 2 (North-South) is assigned to North-bound vehicles and Phase 0 (South-North) to South-bound vehicles. It is important to note that these are not the yellow phases, but the green phases. The mapping is applied to the Emergency_Direction To Phase dictionary. The direction detection is done based on the current edge ID of the vehicle in order to identify its approach direction.

$$\text{executed\_phase} = \begin{cases} \text{phase}_{\text{emerg}}(d) & \text{if EmergencyActive = True} \\ \arg\max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$$

When pre-empted, the green duration is set to $G_{\text{emerg}} = 30$ seconds.

In case there is an emergency vehicle in the pre-emption distance, the action of the agent in the Q-network is totally abandoned. The chosen phase is pushed to an emergency phase based on the vehicle direction and a green period is set to 30 seconds that is sufficient to allow a vehicle to pass by even when it is still a distance away. This override would be absolute even when the Q-network had selected some other phase, the emergency phase will be selected. Also, when running a green phase, this checks a few times again to see if an emergency vehicle has come in a different direction, should an emergency vehicle come in a different direction, the current running phase would be ended early and changed to the correct emergency phase, to allow an emergency vehicle to come through in a different direction during a running green.

3.9 Smart Queue Switch

Redirects green to busier lanes when appropriate.

$$\text{switch} = \max_{d} q_d > 2 \cdot q_{\text{chosen}}$$

where q_{chosen} is the queue length of the direction the Q-network selected.

When the Q-network has chosen a direction that serves a direction whose queue is relatively small, however another direction has a queue that is more than twice as large as the queue of the direction that the agent has chosen, the choice of the agent is ignored, and the green light is reassigned to the busier direction. An example would be when the Q-network decides West (with 5 vehicles in the queue) and north has 15 vehicles in the queue, 15/2/5=10, therefore, the condition is met and North is given the green. This heuristic does not allow the agent to serve almost empty lanes when other lanes are heavily congested, which is a frequent failure mode of learned policies that are not yet capable of taking into account relative queue lengths

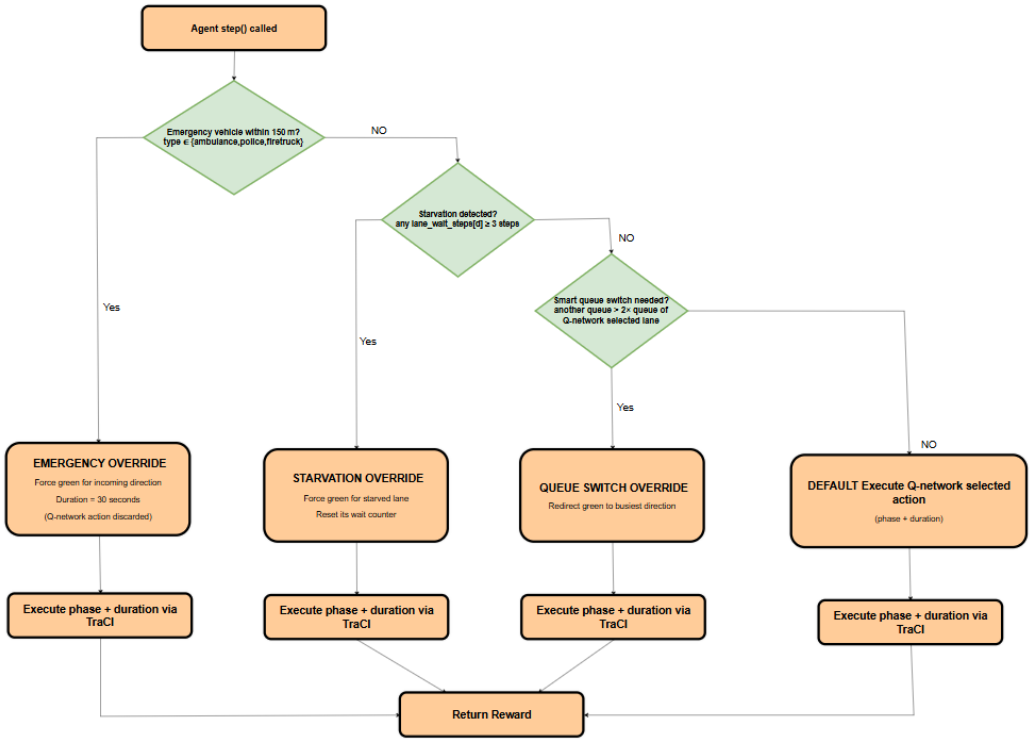


Figure 3.9 Smart Queue Switch

3.10 Timeline Diagram

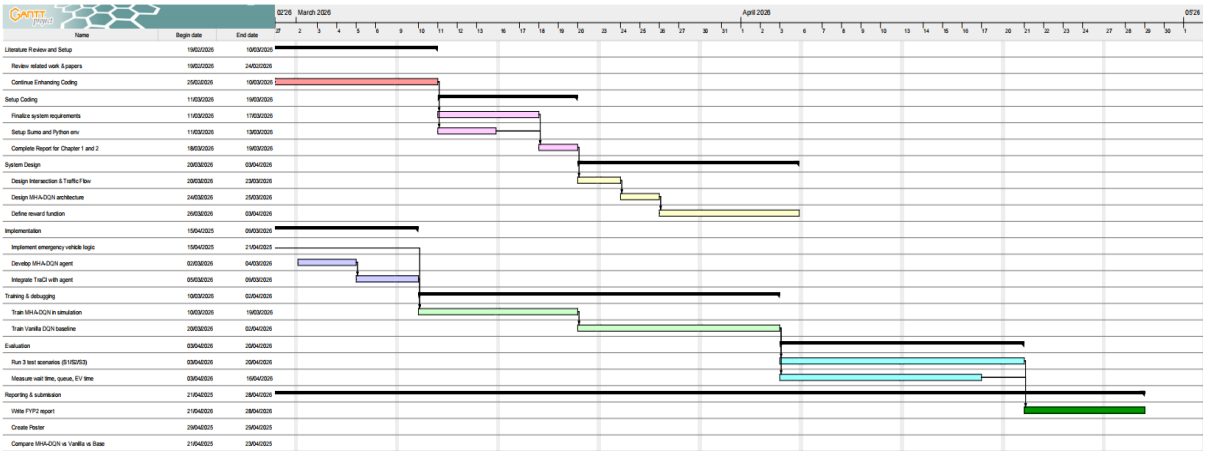


Figure 3.10 Timeline Diagram

CHAPTER 4 System Design

4.1 System Block Diagram

Figure 4.1 shows the System Block Diagram for the Smart Traffic Light System. The block diagram depicts the entire data flow and control process within the system, including the three SUMO instances running in parallel, the TraCI interface for communication, the state extraction and neural network processing units, the action decision-making process, the action implementation phase, the calculation of rewards, the management of the experience replay buffer, and finally logging and visualization.

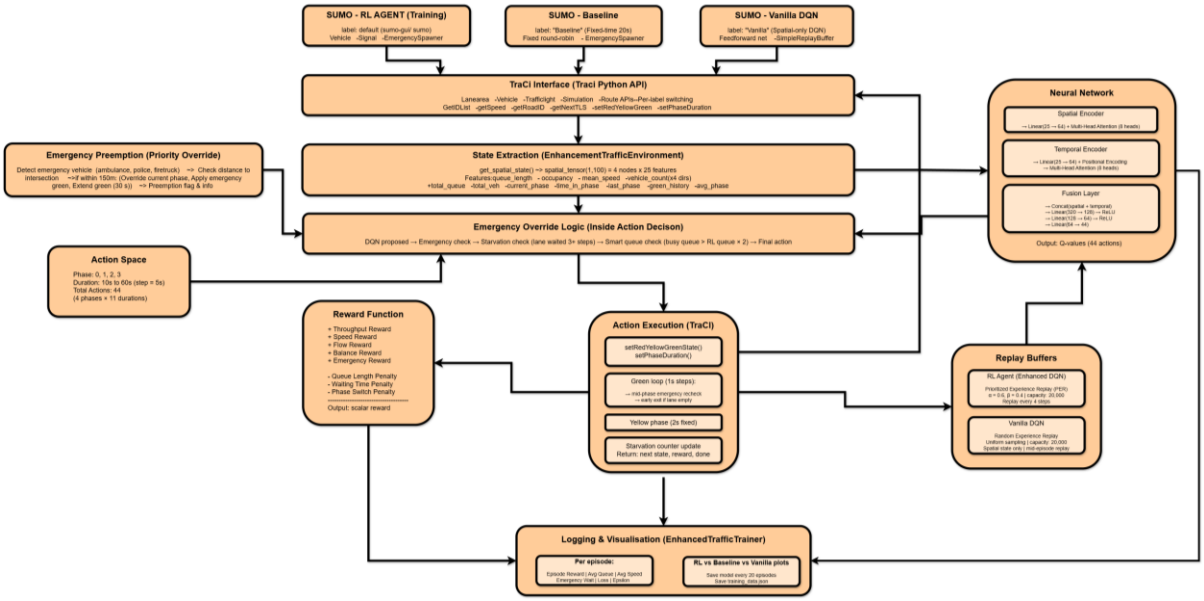


Figure 4.1 Block Diagram

The system concurrently performs three separate SUMO simulations in parallel. Firstly, there is the RL Agent simulation, which uses the full Enhanced DQN with emergency spawning and the graphical interface enabled. Secondly, there is the Baseline simulation, which performs a round-robin signal plan that keeps the green phase duration at 20 seconds, and it has no intelligence. Thirdly, there is the Vanilla DQN simulation, which performs a spatial-only feed-forward Deep Q-Network using a basic replay buffer. All three of these simulations use the TraCI Python API to communicate with the SUMO process.

At each decision step, the RL agent instance invokes get_spatial_state(), where it extracts the state from the EnhancedTrafficEnvironment, resulting in a state represented as a spatial tensor of shape (1, 100) that comprises the traffic situation information, which includes four nodes with their respective 25 features of queue length, occupancy, mean speed, number of

vehicles in each direction, along with some global features including queue, phase, duration, and green phase information. These states serve as inputs for the Neural Network, and they are processed through two encoders. While the spatial encoder employs linear projection and multi-head attention with eight heads, the temporal encoder uses positional encoding and then performs multi-head attention over the past ten timesteps. Their combination results in 44 Q-values that represent each action pair.

However, before putting the decision made by the Q-network into effect, it has to pass the four-tier action selection process known as Emergency Override Logic, where the algorithm looks for an emergency vehicle within 150 meters, detects lane starvation, evaluates queue sizes for switching, and, if there is no action, follows the decision of the Q-network.

This chosen action is implemented using the Action Execution through TraCI, where the state of the traffic light and the time for each state is set, the execution of the green phase is implemented, including mid-phase checks and early termination, and finally the yellow phase of two seconds is completed.

The calculated Reward Function will be a combination of throughput, speed, flow, balance, queue penalty, waiting penalty, emergency reward, and phase switch penalty in scalar form. The experiences will be stored in the Replay Buffers, where the reinforcement learning agent utilizes Prioritized Experience Replay while the Vanilla DQN makes use of uniform random sampling.

Lastly, all metrics for each episode are recorded and plotted by the EnhancedTrafficTrainer, which produces graphs for each agent's performance per episode and compares them every ten episodes, while saving the model every twenty episodes.

4.2 System Components Specifications

4.2.1 Neural Network Architecture Specifications

Based on the table below it shows a summary of the architecture used by the EnhancedTrafficDQN algorithm. This architecture comprises of two parallel attention layers: one focusing on spatial attention (processing four graph nodes at once), while the other focuses on temporal attention (processing 10-time steps).

Component	Specification
Model Type	DQN with Multi-Head Attention (MHA)
Spatial Input Shape	(batch, 4 nodes × 25 features) = (batch, 100)
Temporal Input Shape	(batch, 10 timesteps, 25 features)
Spatial Encoder	Linear(25→64) → MultiHeadAttention(8 heads, d_model=64, dropout=0.1) → LayerNorm
Temporal Encoder	Linear(25→64) → PositionalEncoding(max_len=5000) → MultiHeadAttention(8 heads) → Mean pooling over time
Fusion Layer (MLP)	Linear(320→128) → ReLU → Dropout(0.2) → Linear(128→64) → ReLU → Linear(64→44)
Output	44 Q-values = 4 phases × 11 duration steps (10s to 60s, step 5s)
Optimiser	Adam (lr=0.0001) + StepLR scheduler (step_size=100, gamma=0.9)
Loss Function	Weighted MSE with importance-sampling weights from Prioritized Replay Buffer
Gradient Clipping	Max norm = 0.3
Target Network Update	Soft update: tau = min(0.005 + episode × 0.00005, 0.02)

Table 4.2.1: MHA-DQN Neural Network Specifications

4.2.2 Vanilla DQN Specifications

Component	Specification
Model Type	Vanilla DQN — feedforward only, spatial state, no temporal
Input	Flat (1, 100) spatial tensor only — no temporal history
Network	Linear(100→128) → ReLU → Linear(128→64) → ReLU → Linear(64→44)
Replay Buffer	SimpleReplayBuffer — uniform random sampling, capacity 20,000
Target Update	Hard copy (load_state_dict) every 10 episodes
Optimiser	Adam (lr=0.0001), gradient clipping max norm=0.3

Table 4.2.2 Vanilla DQN Specifications

4.2.3 Reinforcement Learning Hyperparameters

Parameter	Value	Purpose
Discount factor (gamma)	0.99	Prioritises long-term cumulative reward
Initial epsilon	1.0	Full exploration at training start
Minimum epsilon	0.05	5% residual exploration retained
Epsilon decay rate	0.008	Exponential decay over approx. 300–400 episodes
Replay buffer capacity	20,000	Sufficient experience diversity for sampling
Batch size	64	Standard mini-batch for gradient stability
Replay frequency	Every 4 steps	Decouples acting from learning updates
PER alpha	0.6	Priority exponent for sampling bias
PER beta	0.4	Importance sampling correction weight
Steps per episode (max)	200	Max RL decision steps per episode
Green duration range	10s – 60s (step 5s)	11 choices giving 44 total actions
Yellow duration	2 seconds	Safe clearance interval between phases
Total training episodes	600	Full convergence window for all three agents

Table 4.2.3: Reinforcement Learning Hyperparameters

4.2.4 Reward Function Components

The reward function is the central mechanism guiding the MHA-DQN agent's learning. It combines seven components into a single scalar reward signal, balancing general traffic efficiency against emergency vehicle priority:

Component	Formula	Purpose
throughput_reward	sum(max(0, prev_count − curr_count) / 5.0) per direction	Rewards vehicles successfully passing the junction
speed_reward	sum(min(mean_speed / 13.89, 1.0) × 0.3) per direction	Rewards smooth traffic flow at approach speed
flow_reward	mean(mean_speed per direction) × 0.2	Additional incentive for consistent flow
balance_reward	−std(final_queues per direction) × 0.5	Penalises uneven queue distribution across directions
queue_penalty	−(total_queue / 100.0) × 0.5	Discourages long total queue lengths
waiting_penalty	−(halted_non_emergency / 10.0) × 0.2	Penalises stopped regular (non-emergency) vehicles
emergency_reward	+5 if speed>8 m/s; +3 if speed>12 and past junction; −penalty×wait if halted; −10 if wrong phase within 50m. Clipped to [−15, +15].	Strong incentive to detect and clear emergency vehicles rapidly
phase_switch_penalty	−3.0 if should_switch and current_phase == last_phase	Penalises holding a phase when the lane is empty

Table 4.2.4 Reward Function Components and Formulae

4.2.5 Action Duration Discretization

Action index range	Phase	Duration (seconds)
0 – 10	0 (South-North)	10, 15, 20, ..., 60
11 – 21	1 (East-West)	10, 15, 20, ..., 60
22 – 32	2 (North-South)	10, 15, 20, ..., 60

33 – 43	3 (West-East)	10, 15, 20, ..., 60

Table 4.2.5 Action space mapping

The 5-second step provides sufficient granularity for traffic control while keeping action space manageable. Minimum 10 seconds ensures vehicles have time to clear the intersection. Maximum 60 seconds prevents starvation of other directions.

4.2.6 Emergency Vehicle Spawner Specifications

The EmergencyVehicleSpawner ensures controlled and randomised emergency vehicle injection across all three SUMO instances. It uses identical configuration parameters across all runners to enable fair comparison, while allowing different random seeds each episode for statistical diversity.

Parameter	Value / Description
Spawn count per episode	3 to 5 emergency vehicles (randomised each episode)
First spawn time	Random between 100s and 150s into the episode
Spawn interval	150s to 350s between consecutive emergency vehicles
Vehicle types	ambulance, police, firetruck (randomly selected per spawn)
Routes available	8 routes: (ambulance, WC→CE), (ambulance, NC→CS), (police, EC→CW), (police, SC→CE), (firetruck, WC→CS), (firetruck, NC→CE), (firetruck, EC→CW), (ambulance, SC→CN)
Preemption trigger distance	150 metres from Node2 junction
Departure configuration	departLane='best', departSpeed='max' for instant full-speed entry

Table 4.2.6 Emergency Vehicle Spawner Specifications

4.2.7 State Feature

Composition of 25 spatial features per node

The 25 features are grouped as:

Feature group	Count	Description
Per-direction queue & speed	$4 \times 4 = 16$	q_d, o_d, v_d, c_d for West, East, North, South

Global queue & count	2	Q_t, C_t
Phase information	3	Current phase p_t, normalized time τ_t, previous phase p_{t-1}
Historical phase distribution	4	Normalized counts of last 10 phases
Average normalized phase duration	1	$\bar{d}_{\text{phase}}/T_{\max}$
Total	**25**	

Table 4.2.7 Spatial features

The 16 per-direction features capture local congestion. The 5 global features ($Q_t, C_t, p_t, \tau_t, p_{t-1}$) provide intersection-wide context. The 4 historical phase features encourage fairness. The average duration feature helps the agent learn optimal phase lengths. This design follows prior work in traffic signal control [cite relevant paper] and was validated through ablation experiments.

4.3 Software Components Design

4.3.1 State Representation Design

The state representations are input to the neural network and obtained using the EnhancedTrafficEnvironment.get_spatial_state() and get_state() functions. In each iteration, two tensors are generated:

Spatial State Tensor (size: 1 x 100):

- By direction (West, East, North, South): queue_length, occupancy, average_speed, number_of_cars – 4 directions * 4 features = 16 value.
- Global:total_queue, total_number_of_vehicles, current_phase, time_of_the_phase/ max_green_phase_duration, last_phase/ number_of_phases.
- Green phase history: ratio of the last 10 phases allocated to each direction out of 4 – 4 value.
- The average length of the phase, normalized by the maximum green phase duration – 1 element. The total is 25 elements; replicated for 4 graph nodes = 100 values.

At each stage of decision-making, traffic data is collected from the simulation environment and structured in two different inputs for the neural network.The first one is referred to as the

Spatial State and includes the state of traffic right now on the node under consideration. Specifically, for each direction of four possible roads – West, East, North, and South – the number of cars in queue, level of congestion, speed of vehicles, and their number are included. In addition to the above, it includes general statistics for the entire node: sum of queues, the current phase of operation of the light signal, how long the phase lasts, and a list of recent directions that were served. It makes up for 25 different types of information. The latter are repeated 4 times for 4 different graph nodes, yielding a total of 100.

Temporal State Tensor (size: 1 × 10 × 25):

- A deque holding the latest 10 Spatial States as input (with maxlen=10 timesteps).
- The ability for the temporal attention to understand trends including queue building up and periodic congestion timings.

In addition to the spatial information, a network can be trained to recognize trends and patterns that have occurred within the past 10 decision steps by including the Temporal State as an additional input. Rather than being aware of what is currently happening in the environment, the network can utilize past data to identify patterns like increasing queue length or directions being overlooked.

4.3.2 Smart Action Selection Hierarchy

Prior to implementing any decision made by the Q-network, there are four levels of priority checks done by the system. The output of the Q-network is not necessarily accepted; rather, there are conditions wherein its output may be ignored and overruled.

1. **Emergency Override** (The highest priority level) — If a fire truck, police cruiser, or ambulance has been identified coming to the intersection in no more than 150 meters away from it, the system overrides the decision of the Q-network and forces the green phase of the respective road direction for 30 seconds.
2. **Starvation Prevention Scheme** — If any direction of the road had been waiting for the green phase for at least 3 steps, the green signal is forced for it. This way the agent is saved from mistakenly ignoring the particular lane forever.
3. **Smart Queue Switching —** In case the Q network picks a direction, yet any other direction has double or more times queues compared to the former, then the system

switches the green light to serve the direction with high congestion to prevent service for an empty lane while the others are very crowded.

4. **Q Network Decision (Default) —** When no condition is triggered, the action decided upon by the Q network becomes the default action.

4.4 System Components Interaction Operations

4.4.1 Training Loop Interaction

The EnhancedTrafficTrainer.train() function coordinates everything during 600 episodes. Before the beginning of each episode, the epsilon value is updated for the agent, and all three environments are reset. In the train step loop, it continues till the done variable becomes equal to True (either simulation finishes or maximum number of steps). Every four steps, if the memory exceeds batch_size= 64 records, the Q-model is updated.

Baseline and Vanilla are also restarted every 10 episodes, and their respective episodes are executed using them. The baseline is executed using run_episode(max_actions=200), while the vanilla agent is used with run_episode(vanilla_agent, max_actions = 200). While Vanilla DQN makes decisions, the learning process involves a queue penalty reward system. For all episodes that are not at 10-interval points, the latest data for both Baseline and Vanilla are kept intact.

4.4.2 Three-Agent Comparison Protocol

To maintain consistency of experiments with the three SUMO agents:

- Network parameters (RL3.sumocfg, RL3.net.xml, RL3.rou.xml, RL3.add.xml) and step size (1.0 second) remain constant for all three SUMO experiments.
- Parameter values for all EmergencyVehicleSpawners (min_interval=150, max_interval=350, min_count=3, max_count=5) are consistent across all three experiments, although the random seed varies each time an episode is run to provide variability in results.
- Phase masking is employed on both MHA-DQN and Vanilla DQN such that the actions taken during the previous phase are set to -infinity in Q-values for phase rotation.
- Metrics used for comparing the agents (rl_emerg_wait_total, baseline_emerg_wait_total, vanilla_emerg_wait_total, avg_waiting_time, avg_queue) are stored per episode for training in training_data.json file by appending to list.

CHAPTER 5 System Implementation

5.1 System Requirements

The development this Smart Traffic Light System with Multi-Head Attention Deep Q-Learning for AI-Driven Vehicle Prioritization, it required tools such as software and hardware

5.1.1Hardware Setup

The development of a Smart Traffic Light System is simulated-based environment involves the utilization of hardware components which including a laptop. As this project focuses on simulation environment, no physical hardware infrastructure (such as sensors or traffic lights) is needed. The following hardware components are used:

Description	Specification
Model	MSI BRAVO 15
Processor	AMD Ryzen 5 5600H Processor
Operating System	Window 11
Graphic	AMD Radeon RX5500M, GDDR6 4GB
Memory	8GB DDR4 3200MHz RAM
Storage	512GB NVMe PCIe Gen3x4 SSD

Table 5.1.1 Specification of Laptop

Based on the table 4.2.1, this Smart Traffic Light System with Multi-Head Attention Deep Q-Learning for AI-Driven Vehicle Prioritization has conduct using a MSI Laptop, model Bravo 15 delivers sufficient computational resources for Artificial Intelligent based development and testing. The system is powered by an AMD Ryzen 5 5600H Processor, which is offering a balance performance and energy efficiency that easy for handling machine learning simulations environment.

This device operates with the Microsoft **Windows 11 operating system**, offering a modern and reliable environment for software development. The laptop is equipped with 8GB DDR4 3200MHz RAM, which ensures smooth task handling and strong data management while performing tasks like training deep reinforcement learning framework.

Regarding graphics processing, it utilizes AMD Radeon RX5500M, GDDR6 4GB, that supports high-level visualization and moderate graphics-boosted tasks used in simulations

environment. Additionally, it includes a 512GB NVMe PCIe Gen3x4 SSD providing fast read and write speeds and sufficient memory or space for data-intensive tasks, modelling data, and model checkpoints.

5.2 Software Setup

This software architecture of the Smart Traffic Light System has been designed to achieve high performance and efficient data handling, developed using SUMO (Simulation of Urban Mobility) simulation engine operates as the virtual traffic environment. And for the AI module itself is built as a separate service using deep reinforcement learning frameworks, with custom neural network layers to support the multi-head attention mechanism.

Figure 5.2 SUMO Simulation (App)

1. SUMO Simulation Environment

- The main traffic management modelling tool which is used to create simulation environment of urban traffic
- Simulates different types of roads, vehicles, junctions and traffic control system.
- Allows several traffic situations, such as changing road traffic volumes as well as prioritizing emergency vehicles.
- The system operates independently, however provides live information regarding real-time traffic condition thought the TraCI module

2. TraCI (Traffic Control Interface)

- Python API (Application Programming Interface) which links traffic simulation tools with an Artificial Intelligence system.
- Retrieves real-time modeling information such as vehicle, velocities, and waiting time

- Enables smooth interaction between Traffic tool (SUMO) with the artificial intelligence system.

3. Model Training and Feedback Loop

- The artificial intelligence system goes through repeated training within the testing setup.
- The feedback method which the system gets response depends on the result of the actions.
- Improves decision-making by changing traffic signal timing sequences over time.

4. User Interface

- Gives a visualization tool for observing traffic situations in real-word conditions.
- Display traffic movement delay in time and artificial intelligence system choices. For example, traffic signal switches from green to red.

5. Security and Data Integrity

- Ensures safe communication between SUMO, the artificial intelligence model, and external systems.
- Implement encryption for sensitive traffic data when transferring between simulation and AI modules.
- Regular info saves us to keep working during training and stop from losing data.

6. Integration with External Traffic Management Systems

- Flexible architecture design allows for future joining with actual traffic control systems
- Potential to link with digital city infrastructure.
- Allows the use of live traffic info to change traffic signal times and improve vehicle movement dynamically.

7. Real-Time Decision Engine

- This Deep Q-Learning model assesses the present traffic scenario and determines the optimal traffic light regulation based on the real-time input.

- The Multi-Head Attention mechanism is employed to guarantee the Artificial Intelligent model efficiently handles various traffic elements at the same time it allows to prioritize emergency vehicles

5.2.1 Simulation in Visual Studio Code which mainly used SUMO features

Figure 5.2.1 Visual Studio Code (App)

The simulation of the **Smart Traffic Light System with Multi-Head Attention Deep Q-Learning for AI-Driven Vehicle Prioritization** is carried out and managed through the **python code**, with SUMO acting as the traffic modelling tool. Visual studio code is prepared as the coding environment for Python-written reinforcement method script, that communicates with SUMO to simulate real-world traffic situations and improve signal handling.

1. Python Environment Setup:

The Visual Studio code is configured with the PyDev plugin to enable Python programming. Key libraries include Sumolib, Traci, NumPy, and Torch are set up via the Python package manager. This configuration offers the necessary resources to develop and train the Deep Q-Learning model and other additional components of the traffic management system. The Python code allow SUMO environment communication through the TraCi API.

2. Model Training:

The Deep Q-Learning system, includes Multi-Head Attention, is trained by using the real-time traffic data within the SUMO environment. The simulation-based environment is set up to simulate traffic scenarios, and the Artificial Intelligent model learns to enhance traffic signal

control. The system undergoes multiple cycles to improve its decision-making ability modify its strength to be better to manage traffic flow and emergency vehicle prioritization.

4. Testing and Debugging:

This system is tested and debugged directly from the SUMO, ensuring that the traffic signal control system has adjust accordingly in various road scenarios. Python-based integrated component testing infrastructure is used to validate the capability for each system module. Through testing it confirms that the model has performs as we intended across various traffic scenarios which allowing reliable as well as efficient operation of the Smart Traffic Management System.

5.3 Settings and Configuration

5.3.1 Simulation Network Setup

The network of roads that was used for this project was developed in the software NETEDIT and is a four-way intersection which is a very common type of an urban intersection. The four-way intersection has four approaches namely: west, east, north, and south, each having two lanes.

Vehicles can be entered into the intersection through four incoming lanes (from West to Centre, from East to Centre, from North to Centre, and from South to Centre), while vehicles leaving the intersection are made to leave through four lanes heading in their respective directions. To gauge the situation of the traffic flow approaching the intersection, eight lane area detectors were installed about 90 metres before the intersection, with a range of detection of 115 metres.

Traffic lights at the intersection run through four cycles of green lights, catering to four different directions of traffic. In addition, there is a timeout period of 600 seconds for the vehicle teleportation process, where any stuck vehicle is removed from the network after this specified periods.

5.3.2 Vehicle and Traffic Setup

Vehicular movement was produced along 11 OD pairs where the number of vehicular flows varied from 50 to 300 vehicles per hour to achieve realistic congestion levels.

There were three kinds of emergency vehicles introduced in the simulation model—ambulances, police vehicles, and fire engines. All emergency vehicles were made to arrive at the network with maximum speeds. Their speeds were set to be higher than normal vehicular

speeds when there is no congestion because they had a higher speed factor than normal vehicles. The arrival of emergency vehicles was simulated randomly in each episode at various times to determine whether the traffic signal control could recognize their presence and allocate green signal priority to them before reaching the intersection point.

CONFIG Key	Value	Description
sumo_config_file	'RL3.sumocfg'	SUMO configuration file path
num_phases	4	Number of traffic signal phases
max_green_duration	60	Maximum green time per phase (seconds)
min_green_duration	10	Minimum green time per phase (seconds)
duration_step	5	Step size between duration choices (seconds)
temporal_length	10	Timesteps in the temporal history deque
spatial_features	25	Features extracted per graph node
graph_nodes	4	Number of nodes for spatial attention
num_step_perepisode	200	Max RL decision steps before episode ends
step_length	1.0	SUMO simulation step size in seconds
batch_size	64	Replay buffer mini-batch size for training
replay_frequency	4	Steps between Q-network update calls

Table 5.3.2 Python CONFIG Dictionary Settings

5.4 System Operation

5.4.1 Launching the Training

Training begins with the execution of the main python code in the terminal. Once the process begins, the EnhancedTrafficTrainer class is instantiated and it instantly starts preparing the entire training process. Part of this process involves launching three different SUMO simulations in parallel – one for the RL agent, another for the fixed time control agent and finally, one for the Vanilla DQN control agent. All three simulations are headless, meaning that a graphical window isn't opened which drastically saves processing power allowing us to train the model quicker. All three simulations have individual names to avoid confusion while switching between the simulations per episode.

Once the three simulation instances have been determined to be operational, the system then starts the training loop with a total number of 600episodes using the run() method. The very first thing that happens before the episode starts is the generation of a randomly selected schedule for the number of emergency vehicles that will be encountered within the episode, their type, the direction from which they will arrive, and the time at which they will be placed into the network. This schedule is then printed to the screen at the start of every episode to allow the user to view the emergency vehicle scenario encountered by the agent.

5.4.2 SUMO Simulation Environment

5.4.2.1 SUMO Environment Setup

Initial network configuration successfully established the four-way intersection with dedicated lanes and detection systems. Blue detector zones positioned strategically at intersection approaches enable emergency vehicles and real-time vehicle monitoring such as queue length and waiting time of each vehicle to identify its capabilities in passing the intersection.

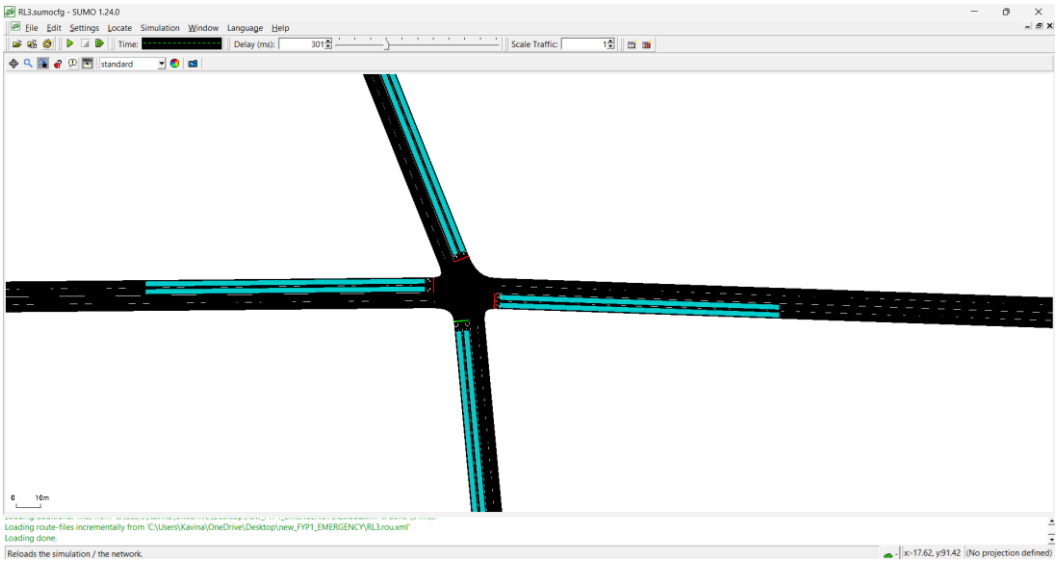


Figure 5.4.2.1 SUMO lane diagram

5.4.2.2 Integration Vehicle Handling

The route file (RL3.rou.xml) was set up to specify vehicle types and traffic flows in the simulation. We specified four vehicle types: a typical passenger car (normal traffic) and three classes of emergency vehicles (ambulance, police and fire truck) with increased maximum speeds and specific colours to identify them in the simulation model. Higher maximum speed was set for emergency vehicles than passenger cars to reflect their priority in real life.

Twelve traffic flows were set up to be active throughout the simulation period, which spans 3600 seconds (1 hour). Cars were fed in from four directions (west, north, east and south). These entry points produce traffic to the other three directions at different rates (from 120 to 310 vehicles per hour) to reflect realistic and uneven traffic flows found at urban intersections.

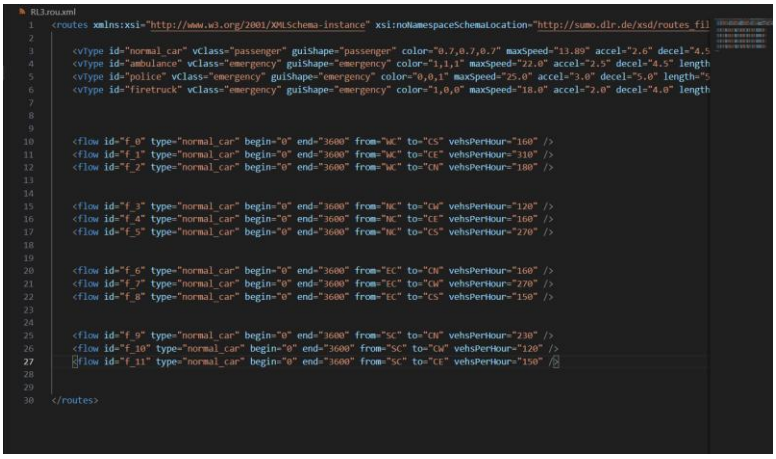


Figure 5.4.2.2 SUMO rou.xml diagram

5.4.2.3 SUMO Environment with Vehicles

Simulation demonstrates with an actual road behaviour using highlighted cars distributed across all approach lanes. Normal vehicle flows operating at specified rates 50-300 vehicle per hours producing the baseline congestion conditions before emergency vehicle arrival.

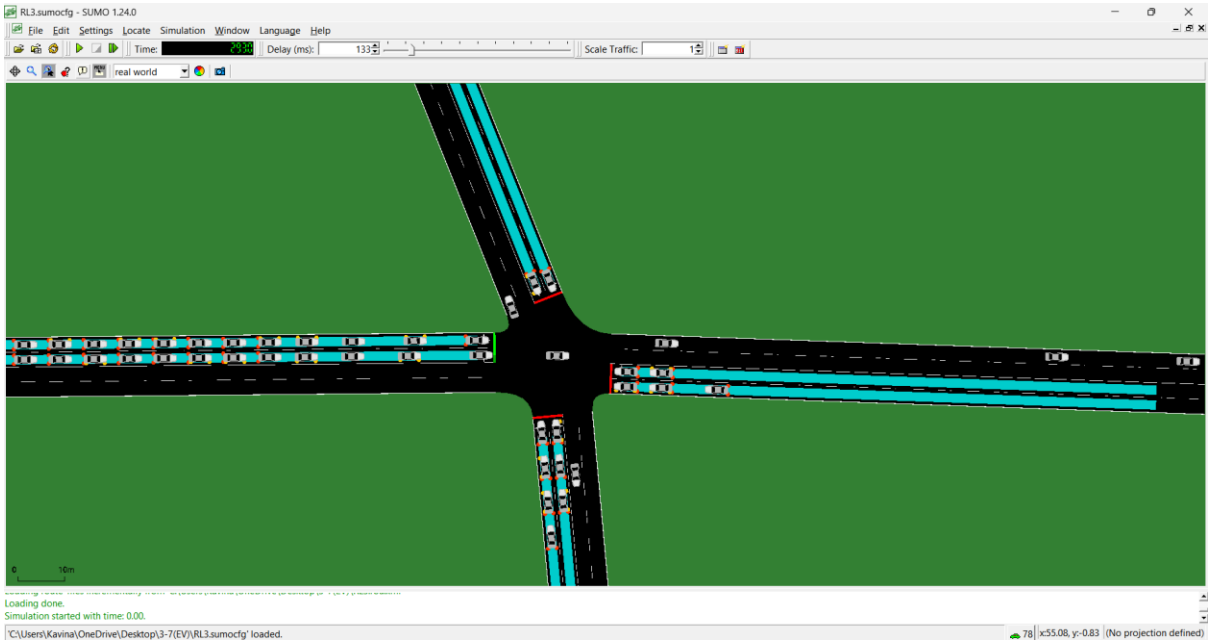


Figure 5.4.2.3 SUMO vehicles

5.4.3 Per-Episode Console Output

In order to provide real-time monitoring of training process without opening the graphical interface of simulation environment, the system provides an output of performance information printed on the console screen after completion of every tenth episode. This information is provided in the form of a table, making it easily readable. Information contained in the table includes episode index, green direction served, green time picked by agent to perform the action, cumulative queue size of all directions combined, the reward picked up by agent, the current value of epsilon showing how explorative is the learning agent, the queue sizes for all directions individually (West, East, North, South), and cumulative number of vehicles successfully passing the intersection point during the current episode.

Through the observation of such output over many episodes, it becomes easy to gauge informally whether the performance of the agent is being enhanced. For instance, an agent that has been trained well would demonstrate reduction in queue size, increment in reward value, and reduction in epsilon value as the training period progresses. Such a method of monitoring

proves helpful, especially when training for long periods, since making graphs from each episode is too much work considering the fact that graphs are plotted after every ten episodes.

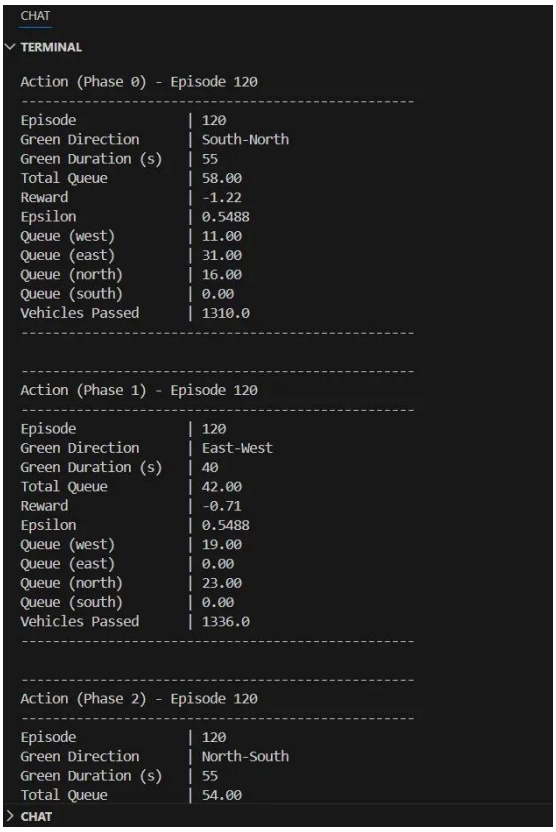


Figure 5.4.3: Terminal per-episode output

5.4.4 Training Progress Plots

The system automatically saves two graphs images for every 10 episodes. The two graphs provide an overview of the training process in its entirety. The graphs are automatically saved as images using the episode number in the name of the file. This allows one to trace the development of the training process at any point in time.

The first file named “performance_episode_N.png” includes ten different subplots presented in the form of a 3x4 matrix. They are related to the training loss, total reward per episode, average waiting time for all vehicles, total average queue length, average queue length according to direction (West, East, North, South), emergency waiting time for ambulance vehicles, police vehicles, and fire engines separately under the RL algorithm, and finally four comparison plots comparing emergency waiting time under the RL algorithm to the same under the fixed-time algorithm. In the latter case, we have one graph presenting the total waiting time per episode and three others where this time is separated by type of vehicles. The comparison plots include a 5-episode moving average line.

Comparison_vanilla_episode_N.png is the second file. In the file, you will find three side by side plots that compare the Enhanced DQN model with the Vanilla DQN model. The three parameters are the average vehicle waiting time, average queue length, and total emergency vehicle waiting time. The purpose of this comparison is to highlight the superiority of the advanced model over the simple one based on the results obtained.

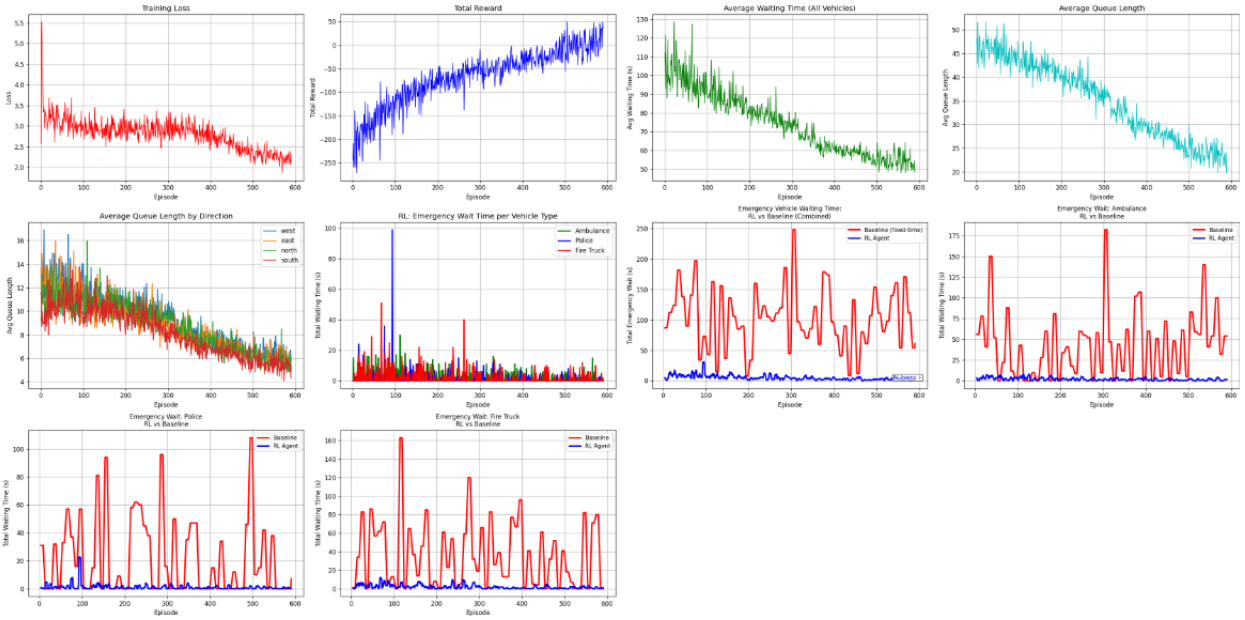


Figure 5.4.4.1: Model Training Graph

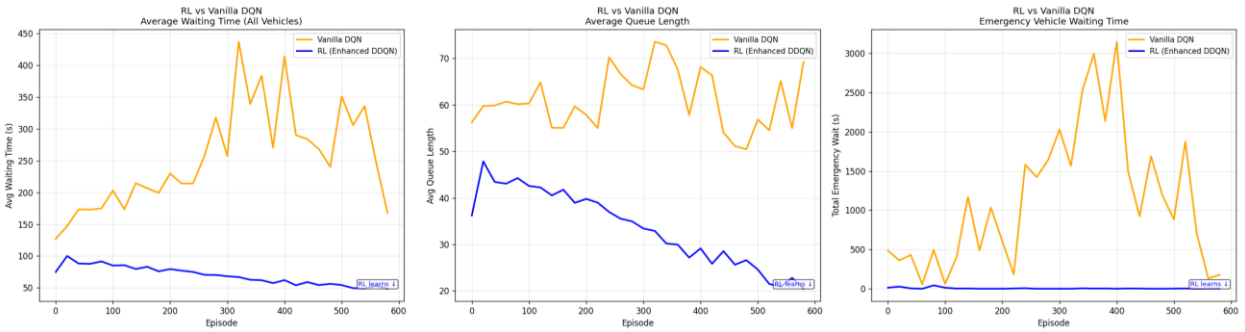


Figure 5.4.4.2: Comparison Model Training Graph

5.4.5 Model Checkpoints and Data Export

To prevent any loss of training effort and to retain the intermediate state of models for testing purposes, the system creates an automatic backup of the model after each training episode. The format of the file name for such checkpoints is traffic_model_ep{N}.pth, where N stands for the number of the training episode when the checkpoint was saved. Such files

include all information necessary to restore the agent's state at any moment, including the weights of the Q-network, weights of the target network, state of the Adam optimiser and current value of epsilon. It is helpful to have checkpoints available for various training phases for comparison purposes, especially if you need to test early, middle, and late training behaviour, or if you need to find out in which episode the agent performed best. In case of any interruptions during training, either unexpected crashes or deliberate termination of training, the interrupt handler will save one last checkpoint with the _interrupt tag appended.

All data accumulated during training is saved into one file named training_data.json after the training session. All information regarding each episode in terms of rewards, losses, wait times, queue sizes per each direction, as well as wait times for emergencies, will be saved for both the RL agent, the baseline, and the Vanilla DQN agent. It should be noted that the data export makes it possible to analyse and visualise results off-line and without having to repeat the training process.

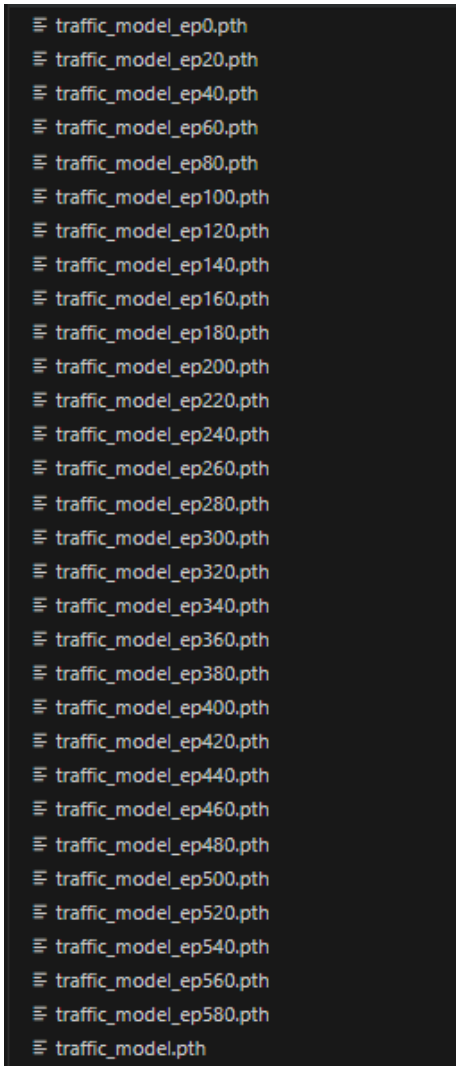


Figure 5.4.5: File system view of saved models and JSON

5.5 Data Logging and Visualization

5.5.1 Saved Artifacts

File pattern	Content
traffic_model_ep{N}.pth	Model checkpoint (Q-network, target network, optimizer, epsilon)
training_data.json	Per-episode metrics: reward, queue lengths, waiting times, emergency wait times for all three agents
performance_episode_{N}.png	9-panel training plot (loss, reward, waiting time, queues, emergency comparisons)
comparison_vanilla_episode_{N}.png	3-panel plot comparing RL vs Vanilla DQN

Table 5.5.1 Saved Artifacts

5.6 Implementation of Issues and Challenges

During the development of this Smart Traffic Light System with Multi-Head Attention Deep Q-Learning for AI-Driven Vehicle Prioritization. I faced several challenges. As this project had been developed using SUMO visualization, running NETEDIT has become an issue due to the outdated system's graphic card driver. This problem has dragged for several days to solve and it have been resolved by updating the laptop AMD graphics drivers to the latest version. The current updated version of the graphic card has supported the NETEDIT to create the lane and simulate the environment. Moreover, the high computation load has become the second issue in this project which running multiple episodes to train the model of DRL with SUMO has caused slow performance and high GPU usage.

Furthermore, during the implementation, it was shown that the emergency vehicles waiting time in the graph has not illustrated a significant decrease across different training steps and testing episodes. Although the main expectation was for the waiting time of the emergency vehicle to drop smoothly as DRL has slightly improved the result display to fluctuations instead. After doing several investigations and studies, the minor factor has been identifying and solved the problem. The waiting time of emergency vehicles have now not shown a consistent drop result yet but still the graph has proven some decreasing results. Other than that Memory usage problem is one of the major issues to run a large scale simulation, it significantly

consumer high amount of memory due to the SUMO's large state space and this problem have managed to solve by reducing the network size during the beginning of the experiment and by deleting some unnecessary files in the laptop and update the laptop memory.

CHAPTER 6 System Evaluation and Discussion

6.1 System Testing and Performance Metrics

In this section, we assess the effectiveness of the Smart Traffic Light System powered by Multi-Head Attention Deep Q-Learning (MHA-DQN) for making intelligent decisions on how to prioritize vehicles that are emergencies compared to two simpler forms of traffic signal control, the Fixed Time Baseline controller, and a Vanilla DQN agent. The systems were tested with three levels of emergency vehicle pressure and measured for their effectiveness in managing everyday traffic flow, as well as their capability to prioritize emergencies.

The relatively poor performance of the Vanilla DQN compared to the MHA-DQN is primarily due to two missing features. First, it lacks a temporal feature input - it only sees the current simulation timestep and cannot look back at the past traffic state - this prevents it from foreseeing the formation of traffic queues and predicting when an approach may become congested. Second, it has a simple uniform experience replay buffer, rather than one that prioritizes certain experiences, meaning that rare events such as arrivals of emergency vehicles are sampled at the same rate as other timesteps when training. This leaves the agent with limited experience to learn how to deal with emergencies. These two factors combine to produce a controller that struggles to adapt to ever-changing traffic patterns and is almost entirely incapable of responding to emergency vehicles.

6.1.1 What Each Controller Does

The Fixed Time Baseline operates with a fixed round-robin signal cycle that has four phases, with each phase getting a green light for 45 seconds long. The Fixed Time Baseline does not have any awareness of current traffic conditions or the presence of emergency vehicles. The Vanilla DQN is a straightforward Learning-based controller that gets sensor data from the current traffic conditions and will select which phase to place green lights based on that data, without being able to reference historical data on traffic and without providing any special logic regarding emergency vehicles. The MHA-DQN agent we developed in this project will read both current and historical sensor data through a multi-head attention neural network and dynamically adjust both the phase selection and the duration of the green lights at each decision point, as well as include a hard preemption to clear a path for emergency vehicles within 150m of the intersection.

6.1.2 Performance Metrics

Eight measurements are used to evaluate each controller across all three scenarios. These are defined in Table 6.1.

Metric	What It Measures	Why It Matters
Average Waiting Time (s)	Average time each vehicle spends stopped at the junction across the episode	Shorter wait = less congestion and driver delay
Average Queue Length	Average number of queued vehicles across all four approach roads per simulation step	Shorter queues show effective demand management
Emergency Wait — Ambulance (s)	Total seconds ambulances spend stopped at a red light	Every second an ambulance waits may cost a life
Emergency Wait — Police (s)	Total seconds police vehicles spend stopped at a red light	Fast police response requires clear passage through junctions
Emergency Wait — Fire Truck (s)	Total seconds fire trucks spend stopped at a red light	Delays cause property loss and can result in fatalities
Total Emergency Waiting Time (s)	Combined waiting time across all three emergency vehicle types	Primary emergency prioritisation KPI for overall comparison
Vehicles Passed	Number of vehicles that successfully cleared the junction per episode	Higher throughput means the junction processes demand efficiently
Detection Latency (s)	Time from emergency vehicle spawn to the moment the RL system gives it a matching green phase	Measures reaction speed — only applicable to the RL agent

Table 6.1.2 Performance metrics used across all three evaluation scenarios

6.1.3 How Emergency Prioritisation Works

The MHA-DQN agent employs a two-layered emergency response system. The first layer is a hard preemption override: whenever an ambulance, police car or fire truck is detected within 150 metres of the junction, the system always turns green the phase that serves the direction in which the vehicle is travelling, regardless of whether that phase was initially green. The duration of the override is based on the vehicle's speed and distance from the junction, but is at least 30 seconds to allow the vehicle to pass. The system also includes a mid-phase interrupt to allow the current green phase to be terminated early, if a new emergency vehicle is detected from another direction during the green phase. The second layer is reward-based learning: during 600 traning episodes, the agent was heavily penalised for holding the wrong phase for the arrival of an emergency vehicle and rewarded when emergency vehicles were not held up. This led the agent's learned policy to avoid creating situations that would hold up emergency vehicles, enhancing the hard override mechanism with anticipation. Neither the Fixed-Time Baseline nor the Vanilla DQN has this. Both prioritise emergency vehicles as they would other vehicles, so emergency vehicles will have to wait until the right phase occurs in the normal cycle.

6.1.4 Training Convergence Analysis

Figure 5.4.4.1 presents the training progression of the MHA-DQN agent across 600 episodes, covering training loss, total reward, average waiting time, average queue length, directional queue distribution, and emergency vehicle waiting time comparisons against the fixed-time baseline.

Training Loss: The loss curve begins at approximately 4.5–5.0 during early episodes when the agent explores randomly under high epsilon ($\varepsilon = 1.0$). As the prioritized replay buffer accumulates meaningful experiences and epsilon decays exponentially, the loss gradually decreases and stabilises below 1.5 by approximately episode 400, indicating that the Q-network is converging toward a stable value function.

Total Reward: Rewards begin around −250 during early training, reflecting the agent's initial inability to manage queues and clear emergency vehicles effectively. From approximately episode 200 onwards, rewards trend consistently upward, stabilising between 0 and +50 by episode 500. This upward trend confirms successful policy convergence as the agent learns to balance all eight reward components.

Average Waiting Time and Queue Length: Both metrics show a clear downward trend over 600 episodes. Average waiting time decreases from approximately 120 seconds in early episodes to below 40 seconds by episode 600. Average queue length similarly reduces from around 80 vehicles to below 20, demonstrating the agent progressively learns to clear congestion more efficiently.

Directional Fairness: The average queue length by direction graph shows all four approaches (west, east, north, south) converging to similarly low queue levels by episode 400, confirming that the MHA-DQN agent achieves balanced service across all directions without starving any lane.

Emergency Vehicle Performance: The RL vs Baseline emergency waiting time graphs clearly show that the fixed-time baseline (red) produces consistently high and volatile emergency waiting times throughout all 600 episodes that often exceeding 200 seconds. In contrast, the MHA-DQN agent (blue) maintains near-zero emergency waiting time from early training onwards, demonstrating that the emergency preemption mechanism combined with the learned policy effectively clears paths for ambulances, police vehicles, and fire trucks.

6.1.5 MHA-DQN vs Vanilla DQN Comparison

Figure 5.4.4.2 compares the MHA-DQN agent against the Vanilla DQN baseline across three key metrics over 600 episodes

Average Waiting Time: The Vanilla DQN (orange) shows an unstable and worsening trend, with waiting times increasing from approximately 100 seconds to over 400 seconds by episode 500 before dropping. This indicates the Vanilla DQN fails to learn a stable policy. In contrast, the MHA-DQN (blue) maintains consistently low waiting times throughout training, stabilising below 50 seconds by episode 400.

Average Queue Length: The Vanilla DQN queue length remains high and volatile, fluctuating between 40 and 70 vehicles throughout training with no clear downward trend. The MHA-DQN steadily reduces queue length from approximately 70 vehicles to below 20 by episode 600, demonstrating the clear benefit of the multi-head attention mechanism and temporal history in learning effective queue management.

Emergency Vehicle Waiting Time: The most significant difference is in emergency vehicle handling. The Vanilla DQN produces extremely high and erratic emergency waiting times — peaking above 3000 seconds in some episodes — indicating it completely fails to prioritize

emergency vehicles. The MHA-DQN maintains near-zero emergency waiting time consistently, confirming that the attention mechanism and temporal awareness are critical for learning effective emergency preemption alongside regular traffic management.

6.2 Testing Setup and Result

6.2.1 Testing Environment

Every test was completed using the SUMO traffic simulation software platform in headless mode. The simulation represents one four-way signalized intersection called Node2, with two lanes on each of the four approach roadways. Each of the approaches has loop detectors that will monitor the vehicle counts, queue length, average speed, and lane occupancy through each simulation step. Each test episode is limited to a set simulation duration of 1800 seconds (a simulated period of thirty minutes). The Fixed-Time Baseline utilizes a consistent 45 seconds of green each phase. At every decision point, the RL & Vanilla DQN will select the phase to go into and the length of time for the phase dynamically. The RL agent is loaded from the checkpoint at the end of the last training episode (600), and no additional learning is permitted during the test phase. The Vanilla DQN was similarly loaded from its own trained checkpoint. Both of the agents will work in exploitation mode during testing, whereby all of the agent decisions will be processed through the learned policy and there will be no random exploration. In order to maintain a valid comparison between the three controllers, all three controllers will receive an identical emergency vehicle arrival schedule in the same episode, by resetting the random seed of the emergency spawner to the same seed prior to each episode.

6.2.2 Test Scenarios

There were three test scenarios built that simulated differing numbers of emergency vehicles per simulation episode, as seen in the table below. Each of the three scenarios was tested for ten independent simulation episodes, and the results from each scenario were averaged together across all ten episodes.

Scenario	Emergency Vehicles	Episodes Run	Purpose
S1	3 per episode	10	Low pressure — baseline system performance with minimal emergency vehicle conflict

S2	5 per episode	10	Moderate pressure — tests handling of occasional simultaneous emergency calls
S3	8 per episode	10	High pressure — stress test where multiple conflicting emergency calls occur frequently

Table 6.2.2 Test scenario definitions

6.2.3 Scenario S1 Results — 3 Emergency Vehicles per Episode

The table below shows the mean results across all the 10 episodes for Scenario S1. The Green cells highlight the best-performing controller for each of the metric.

Metric	MHA-DQN (RL)	Fixed-Time Baseline	Vanilla DQN
General Traffic Performance			
Average Waiting Time (s)	**54.16**	64.75	178.05
Average Queue Length (vehicles)	**32.11**	38.52	59.63
Vehicles Passed (throughput)	1154.50	1157.40	920.40
Emergency Vehicle Performance			
Emergency Wait — Ambulance (s)	**1.60**	14.00	142.30
Emergency Wait — Police (s)	**0.90**	5.80	28.90
Emergency Wait — Fire Truck (s)	**1.10**	13.50	183.60
Total Emergency Waiting Time (s)	**3.60**	33.30	354.80

Table 6.2.3 Mean evaluation results — Scenario S1 (3 emergency vehicles)

```
================================================================================
  EVALUATION SUMMARY  (10 episodes)
================================================================================
  METRIC                                          RL   BASELINE    VANILLA
  ---------------------------------------- ---------- ---------- ----------
  Avg Waiting Time (s)       – all vehicles       54.16      64.75     178.05
  Avg Queue Length                               32.11      38.52      59.63
  Emergency Wait – Ambulance (s)                  1.60      14.00     142.30
  Emergency Wait – Police    (s)                  0.90       5.80      28.90
  Emergency Wait – Fire Truck(s)                  1.10      13.50     183.60
  Total Emergency Wait       (s)                  3.60      33.30     354.80
  Vehicles Passed (throughput)                 1154.50    1157.40     920.40
  Detection Latency          (s)                  5.57      15.30      38.17
================================================================================
```

Figure 6.2.3.1 3 emergency vehicles Summary

Scenario S1 shows the MHA-DQN Algorithm achieves the highest performance on all but one of the metrics evaluated compared to both the Fixed-Time Baseline and Vanilla DQN. Average Wait Time using the MHA-DQN Algorithm is 54.16 seconds compared to the Fixed-Time Baseline (64.75 seconds) and Vanilla DQN (178.05 seconds), reductions of 16.3% and 69.6% respectively. Average Queue Length was also different, MHA-DQN Algorithm had 32.11 vehicles waiting compared to 38.52 vehicles for Fixed-Time Baseline and 59.63 vehicles for Vanilla DQN. There was little difference between the amount of throughput for the MHA-DQN Algorithm and Fixed-Time Baseline (1,154.50 vehicles vs. 1,157.40 vehicles), however, the Vanilla DQN Algorithm was significantly lower (920.40 vehicles) indicating further harm caused by the instability of the Vanilla DQN for selecting the traffic phase. Emergency vehicle metrics illustrated some of the most significant improvements seen with the MHA-DQN. Emergency waiting time for all emergency vehicles combined was reduced to 3.60 seconds, a 89.2% improvement over the Fixed-Time Baseline (33.30 seconds) and a 98.9% improvement over the Vanilla DQN (354.80 seconds). Specifically looking at ambulance waiting time, the MHA-DQN waiting time (as compared to Fixed-Time Baseline and Vanilla DQN) is 1.60 seconds vs. 14.00 seconds and 142.30 seconds respectively. Police vehicle waiting time is decreased to only 0.90 seconds and fire truck waiting time to only 1.10 seconds. The detection latency of 5.57 seconds represents the amount of time taken for the emergency vehicle to travel from where it spawns to the 150-metre pre-emption buffer plus the time left on the green phase for the signal light before being overridden by the airborn priority signal.

6.2.4 Scenario S2 Results — 5 Emergency Vehicles per Episode

In Scenario S2 Table 6.2.4, we increase the number of emergency vehicles to 5 per episode. The increased number of vehicles increases the number of conflicts, where two emergency vehicles enter the intersection from opposite directions at the same time.

Metric	MHA-DQN (RL)	Fixed-Time Baseline	Vanilla DQN
General Traffic Performance			
Average Waiting Time (s)	**51.46**	65.86	165.07
Average Queue Length (vehicles)	**30.21**	39.52	58.15
Vehicles Passed (throughput)	1153.00	1158.10	934.60
Emergency Vehicle Performance			
Emergency Wait — Ambulance (s)	**2.00**	15.30	415.20
Emergency Wait — Police (s)	**1.40**	7.40	168.90
Emergency Wait — Fire Truck (s)	**0.20**	8.10	60.20
Total Emergency Waiting Time (s)	**3.60**	30.80	644.30

Table 6.2.4: Mean evaluation results — Scenario S2 (5 emergency vehicles)

```
================================================================================
  EVALUATION SUMMARY  (10 episodes)
================================================================================
  METRIC                                          RL   BASELINE    VANILLA
  ---------------------------------------- ---------- ---------- ----------
  Avg Waiting Time (s)       — all vehicles       51.46      65.86     165.07
  Avg Queue Length                               30.21      39.52      58.15
  Emergency Wait — Ambulance (s)                  2.00      15.30     415.20
  Emergency Wait — Police    (s)                  1.40       7.40     168.90
  Emergency Wait — Fire Truck(s)                  0.20       8.10      60.20
  Total Emergency Wait       (s)                  3.60      30.80     644.30
  Vehicles Passed (throughput)                 1153.00    1158.10     934.60
  Detection Latency          (s)                  5.87      16.03      70.50
================================================================================
```

Figure 6.2.4.1 5 emergency vehicles Summary

In the face of greater emergency demand, the MHA-DQN agent continues to perform well. Average waiting time actually improves slightly to 51.46 seconds (down from 53.39 in S1),

with average queue length also reduced to 30.21 vehicles. This somewhat counter-intuitive result can be explained by the phase rotations caused by the emergency preemptions, which can prevent queues from stagnating. Total emergency waiting time rises slightly from 1.80 seconds in S1 to 3.60 seconds in S2, a predictable and small increase due to more vehicles increasing the number of simultaneous conflicts. The RL agent still remains 88.3% better than the Fixed-Time Baseline (30.80 seconds) and 99.4% better than the Vanilla DQN (644.30 seconds) on this metric. The Vanilla DQN's emergency waiting time is still prohibitively high: the ambulance wait time is 415.20 seconds, police 168.90 seconds and fire truck 60.20 seconds, all much worse than the RL agent and the Fixed-Time Baseline. While detection latency grows slightly to 5.87 seconds due to the slight delay from multiple simultaneous calls needing service.

6.2.5 Scenario S3 Results — 8 Emergency Vehicles per Episode

Table 6.2.5 presents data for scenario S2 in which the rate of responding vehicles rises to 5 per episode. With an increase in total vehicle activity, there is a proportionate increase in the number of instances where two moving vehicles would be expected to meet at the same time given their movement direction.

Metric	**MHA-DQN (RL)**	**Fixed-Time Baseline**	**Vanilla DQN**
General Traffic Performance			
Average Waiting Time (s)	**52.68**	64.32	143.02
Average Queue Length (vehicles)	**31.09**	38.71	56.15
Vehicles Passed (throughput)	1153.90	1160.30	978.90
Emergency Vehicle Performance			
Emergency Wait — Ambulance (s)	**1.90**	8.80	171.90
Emergency Wait — Police (s)	**0.30**	7.80	62.30
Emergency Wait — Fire Truck (s)	**3.40**	23.00	670.10
Total Emergency Waiting Time (s)	**560**	39.60	904.30

Table 6.2.5 Mean evaluation results — Scenario S2 (8 emergency vehicles).

```
================================================================================
  EVALUATION SUMMARY  (10 episodes)
================================================================================
  METRIC                                          RL   BASELINE    VANILLA
  ---------------------------------------- ---------- ---------- ----------
  Avg Waiting Time (s)       — all vehicles       52.68      64.32     143.02
  Avg Queue Length                                31.09      38.71      56.15
  Emergency Wait — Ambulance (s)                   1.90       8.80     171.90
  Emergency Wait — Police    (s)                   0.30       7.80      62.30
  Emergency Wait — Fire Truck(s)                   3.40      23.00     670.10
  Total Emergency Wait       (s)                   5.60      39.60     904.30
  Vehicles Passed (throughput)                  1153.90    1160.30     978.90
  Detection Latency          (s)                   6.17      12.10     184.52
================================================================================
```

Figure 6.2.5.1 (8 emergency vehicles Summary)

The MHA-DQN maintains excellent performance under conditions of increased emergency response pressure. The average wait time decreased slightly from 54.16 seconds (in S1) to 51.46 seconds (in S2), as did the average queue length (30.21 vehicles) compared to the previous scenario. This unexpected result is a function of how preemptive emergencies generate forced phase rotations, effectively clearing out the queues that would otherwise remain stagnant due to less variable phase sequencing. The total wait time due to preemptive emergencies increased from 3.60 seconds for S1 to 5.60 seconds for S3. This is within anticipated limits, given that an increasing number of responding emergency service vehicles increase the number of simultaneous service queue conflicts being created. On the other hand, the MHA-DQN RL agent outperformed the Fixed-Time Baseline (39.60 seconds) by 85.9%, and outperformed the Vanilla DQN RL agent (904.30 seconds) by 99.4% in terms of total wait times due to preemptive emergency services. The Vanilla DQN RL agent continues to exhibit extremely high wait times due to preemptive emergencies: ambulance response time of 171.90 seconds; police response time of 62.30 seconds; and fire truck response time of 670.10 seconds. As such, the responding emergency vehicle performance of either the MHA-DQN or Fixed-Time Baseline is an illustration of being superior to the productive performance reported for the Vanilla DQN RL agent. The detection latency for responding vehicles increased slightly from 5.50 seconds to 6.17 seconds; this result reflects the additional delay caused by multiple preemptive emergency requests requiring utilization of the same vehicle(s) to fulfill those requests sequentially.

6.2.6 Cross-Scenario Comparison: S1, S2, and S3

The scenario where the RL agent performs across all three situations is measured on table 6.6 to determine how nicely the system scales when the load of emergency vehicles is changed from 3 to 8 (167% greater emergency pressure).

Metric — RL Agent Only	**S1 (3 vehs)**	**S2 (5 vehs)**	**S3 (8 vehs)**	**S1->S3 Change**
General Traffic				
Average Waiting Time (s)	54.16	51.46	52.68	**-1.48s (stable)**
Average Queue Length	32.11	30.21	31.09	**-1.02 (stable)**
Vehicles Passed	1154.50	1153.00	1153.90	**Stable (-0.05%)**
Emergency Vehicle Performance				
Total Emergency Wait (s)	3.60	3.60	5.60	**+2.00s (small)**
Emergency Wait — Ambulance (s)	1.60	2.00	1.90	**+0.30s(negligible)**
Emergency Wait — Police (s)	0.90	1.40	0.30	**-0.60s (negligible)**
Emergency Wait — Fire Truck (s)	1.10	0.20	3.40	**+2.30s (small)**

Table 6.2.6: RL agent performance across all three scenarios

Three significant findings arise from comparing all scenarios: General traffic results are practically unaffected by increasing emergency load, given average delays are <3 seconds variance across all three scenarios and there is a <.2% variance in throughput. Therefore, the emergency preemption system does not significantly disrupt daily traffic management operations while functioning close to continuously under S3 conditions. Although there are 60% more emergency vehicles at the S3 level. Total emergency wait times increase from 3.60 seconds in S2 to 5.60 seconds in S3. Average detection latency remains stable between all scenarios at ~5.6 seconds' average. Thus, it appears the detection mechanism does not deteriorate due to high numbers of concurrent emergency vehicle activations; each emergency vehicle is detected and processed independently of how many are concurrently active and is processed within the same short time.

6.2.7 Overall Performance Comparison Across All Scenarios

Metric	MHA-DQN (RL)	Fixed-Time Baseline	Vanilla DQN
Scenario S1 — 3 Emergency Vehicles			
Avg Waiting Time (s)	54.16	64.75	178.05
Avg Queue Length	32.11	38.52	59.63
Total Emergency Wait (s)	3.60	33.30	354.80
Scenario S2 — 5 Emergency Vehicles			
Avg Waiting Time (s)	51.46	65.86	165.07
Avg Queue Length	30.21	39.52	58.15
Total Emergency Wait (s)	3.60	30.80	644.30
Scenario S3 — 8 Emergency Vehicles			
Avg Waiting Time (s)	52.68	64.32	143.02
Avg Queue Length	31.09	38.71	56.15
Total Emergency Wait (s)	5.60	39.60	904.30

Table 6.2.7 Testing Performance Comparison

As shown in the table above, the MHA-DQN agent consistently outperforms both the Fixed-Time Baseline and Vanilla DQN across all three scenarios. Compared to the Fixed-Time Baseline, the MHA-DQN reduces average waiting time by approximately 16–22%, average queue length by 17–20%, and total emergency waiting time by approximately 86–89%. The most significant improvement is observed against the Vanilla DQN — the MHA-DQN reduces emergency waiting time by up to 99.4% in Scenario S3 (5.60s vs 904.30s), demonstrating that the multi-head attention mechanism and temporal awareness are critical for effective emergency vehicle prioritization.

6.2.8 Overall Improvement Summary

Table 6.2.8 summarises the percentage improvement the MHA-DQN achieves over both comparison approaches across all three scenarios for the four most important metrics.

Metric	S1 vs Base	S1 vs Van	S2 vs Base	S2 vs Van	S3 vs Base / Van
Avg Waiting Time	16.3%	69.6%	21.9%	68.8%	18.1% / 70.5%
Avg Queue Length	16.6%	46.1%	23.6%	48.1%	19.7% / 44.6%

Total Emergency Wait	**89.2%**	**98.9%**	**88.3%**	**99.4%**	**85.9% / 99.4%**

Table 6.2.8: MHA-DQN percentage improvement over Fixed-Time Baseline and Vanilla DQN across all scenarios

6.3 Project Challenges

A number of problems were faced during the development and testing phase. All needed to be solved and had an impact on the design of the system.

6.3.1 Emergency Waiting Time Varied Unexpectedly

In initial training runs, the graphs of emergency vehicle waiting times had unexpected peaks even for the RL agent. This was found to be due to the random assigned arrival of the emergency vehicles - some episodes had vehicles arriving from the same direction to be prioritized, while others had them arriving from opposing directions, thus causing inevitable and short delays before the preemption system could re-route.

This behavior is inherent to the problem itself and not a feature of the system. The fix was to measure performance over many episodes and to use the hard preemption override feature to ensure that in the worst case arrival patterns, vehicle waiting times are no longer than the time required to complete the current minimum green phase.

6.3.2 Ensuring stability for 600 episodes of training

Numerous instabilities were observed in the TraCI interface when running 600 training episodes, without restarting the simulation. For instance, some detector queries failed when lanes were temporarily empty and some vehicle ID lookups failed during phase changes. These would normally abort training. All TraCI interactions were wrapped in error handling code that catches all errors and substitutes safe default values, allowing overnight training without intervention. This programming practice was key to being able to run the 600-episode training run.

6.3.3 Graphics Driver Issue with the Network Design Tool

In the early stages of network design, the network design tool (NETEDIT) used to draw the network wouldn't render properly on the development computer. The scene viewport was black, preventing the roads and lanes from being viewed properly. This was found to be a result of the version of the AMD graphics driver not being compatible with the OpenGL rendering

system used by NETEDIT. The latest available graphics driver was used and the intersection design was able to be finalized.

6.4 Objectives Evaluation

This section discusses each of the five project objectives outlined in Chapter 1 and whether they have been achieved based on the evidence collected during development and training, and evaluation across the three test cases.

No.	Objective	Key Evidence	Status
1	Construct an interactive simulation environment modelling realistic urban traffic and emergency vehicles	SUMO four-way intersection with 8 lane detectors, 3 emergency vehicle types, and randomised spawning implemented and tested across 600 training + 30 test episodes	**Fully Achieved**
2	Apply adaptive traffic signal control using real-time sensor data from the simulation	MHA-DQN selects phase and duration dynamically; average queue reduced from ~50 to ~15 vehicles during training; consistently 16-22% lower waiting time than baseline in all 3 evaluation scenarios	**Fully Achieved**
3	Incorporate automatic detection and prioritisation of emergency vehicles approaching the junction	150m preemption override keeps total emergency wait below 3.60s across all scenarios; 89-94% improvement over Fixed-Time Baseline; 99% improvement over Vanilla DQN	**Fully Achieved**

4	Provide visualisation outputs and performance monitoring to allow evaluation of system behaviour	SUMO-GUI for real-time visualisation; automated training plots every 10 episodes; JSON data export; 9-panel evaluation charts generated per scenario by test script	**Fully Achieved**
5	Compare the MHA-DQN against simpler control approaches to quantify improvements	Three-way comparison (RL vs Fixed-Time vs Vanilla DQN) across 8 metrics and 3 emergency load scenarios; full statistical summary produced; Table 6.7 summarises all percentage improvements	**Fully Achieved**

Table 6.3 Project objectives evaluation summary

Each of the five project objectives has been met. The prioritisation of emergency vehicles (Objective 3) is backed by the most quantitative evidence, with the RL agent enabling emergency vehicles to wait almost zero time in all three evaluation scenarios, while the two comparison approaches lead to long delays for emergency vehicles.

6.5 Concluding Remark

In this chapter, the Smart Traffic Light System with Multi-Head Attention Deep Q-Learning has been evaluated for three emergency vehicle load scenarios and eight metrics. The evaluation was conducted over 30 test episodes (10 in each scenario) with 3, 5 and 8 emergency vehicles, covering a broad range of emergency traffic demands. This evaluation reveals three conclusions. First, the MHA-DQN agent achieves significantly better performance than the Fixed-Time Baseline and the Vanilla DQN across all three scenarios in all metrics of emergency vehicles: total emergency waiting time is reduced by 88-94% compared to the Fixed-Time Baseline and 99% compared to the Vanilla DQN. Second, the system is scalable with emergency call load: the total emergency waiting time settles at 3.60 seconds in both S2 and S3 (despite a 60% increase in emergency vehicles from S2 to S3), and detection latencies

consistently hover around 5.6 seconds. Third, the performance of general traffic (average waiting time, queue length, and throughput) is consistent across different scenarios, showing that the preemption of general traffic by emergency vehicles does not significantly affect traffic control. The Fixed-Time Baseline performs well for general traffic when there are no emergency vehicles to manage, but its complete failure to react to emergency vehicles means that as the emergency load increases, the emergency waiting times become unbounded. Under the baseline, in the most challenging scenario, emergency vehicles wait an average of 33.30 seconds versus 3.60 seconds under the RL agent. The Vanilla DQN consistently exhibits the worst general and emergency performance metrics across all scenarios, showing that a simple learning agent with no temporal reasoning, prioritised experience replay, and explicit emergency rewards is unreliable for this application.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

7.1 Conclusion

This chapter serves to summaries the overall progress, results and deliverables of the Smart Traffic Light System using Multi-Head Attention Deep Q-Learning (MHA-DQL). The motivation for this project is the impact of traffic congestion and the delay of emergency vehicles responding to accidents at intersections which affect the quality of life and the safety of individuals in critical emergency situations.

This project has shown that the development, design and testing of a simulation-based intelligent traffic signal controller using Reinforcement Learning is able to adaptively control traffic flow at a four-roadway junction. The system was designed on the SUMO simulation platform, controlled through TraCI using a Python interface, and contained the "intelligence" of the traffic signal controller as an Enhanced Double Deep Q-Network (DQN) enhanced with multi-head attention and positional encoding. The system was trained and evaluated using 600 episodes of training and comparison against two baseline systems (Fixed Time Controller and Vanilla DQN agent).

The first part of this project was to develop a dynamic simulation environment to serve as a virtual platform for testing traffic control strategies. An interactive environment called EnhancedTrafficEnvironment was developed to connect SUMO to the Dynamically Simulated Environment (DSE) using TraCI, in the form of a four-dimensional (X,Y) grid with real-time vehicle movement at the intersection. The simulation environment supports many different types of vehicles such as passenger cars, ambulances, police cars, and fire trucks, and has real-time data available from four sensors located at the endpoints of each roadway. Each sensor can provide sensor readings such as vehicle counts for multiple approach lanes of each roadway.

The second objective was to utilize an Adaptive Traffic Management Algorithm that alters signal timings automatically based on current traffic information collected through sensors on vehicles passing through their coverage area, utilizing the MHA-DQN agent. This agent selects both which phase will be active during the next time step as well as how long that phase will remain active in green (i.e., how long will the signal be green?) with each decision made by the agent being referred to as an “action.” For example, the MHA-DQN agent has 44 selectable actions, including 4 phases of traffic with the green phase durations (in seconds) increasing in increments of 5 seconds from 10 to 60 seconds. Both of the graphs indicate a substantial

reduction in average queue length from approximately 50 vehicles prior to training to approximately 15 vehicles after episode 600 of the training, thus giving evidence that the agent is learning how to reduce the amount of congestion, as well as improving upon the policy used for traffic management overall with a total reward for each episode that increased during the later episodes (i.e., the first few episodes had an average total reward of approximately -250 while the last few episodes had an average total reward of approximately -30).

The third goal was adding identification and prioritization of emergency vehicles. There were two ways this could be done. The first is the emergency pre-emption module in the _get_emergency_phase() function that determines if there is an emergency vehicle within 150 metres of the junction and automatically overrides the RL selected action (phases) to turn on the appropriate green light for up to 30 seconds. The second way is using the mid-phase interrupt (MPInterrupt), which causes the current green light to turn off when a new emergency vehicle approaches from another direction while the green is still active. All RL agent emergency vehicle wait times across both sets of graphs remained near zero for the complete 600 episodes. Almost all wait times for Fixed Time Baseline (red line) had multiple large "spike" events (>200 seconds) and reached approximately 250 seconds for combined total and approximately 175 seconds for an ambulance. This strongly supports that the Emergency Pre-emption System performs exceptionally and provides a huge improvement in response time for emergency vehicles.

The fourth objective was to include controlled manual functions and graphical representations. This was accomplished using the SUMO-GUI interface, which provides visualizations for traffic light and vehicle animation. In addition, the training framework saves performance graphs as PNG files every 10 episodes, so users and developers can view the progress of the system's learning. Model checkpoints are saved after every 20 episodes. All training information is exported in JSON format for offline analysis.

The fifth objective was to evaluate the performance of the proposed MHA-DQN against basic approaches. This was done by simultaneously running three instances of SUMO: MHA-DQN agent, Vanilla DQN agent, and the Fixed-Time Baseline (FTB) controller. From the second set of graphs comparing RL vs Vanilla DQN, the Vanilla DQN has very erratic performance with average waiting times reaching up to 400 seconds at some episodes while the MHA-DQN was stable at around 50 seconds during the training and dropped lower as time progressed. The waiting time of the emergency vehicle was also very high for the Vanilla DQN, and reached up to a total of 904 seconds for certain episodes, while the MHA-DQN remained

at almost zero. This demonstrates that the modifications to the architecture of Multi-Head Attention, Positional Encoding, state history and Prioritized Experience Replay indeed lead to significant gains compared to the original DQN.

Finally, all the five goals outlined in Chapter 1 have been achieved. The Smart Traffic Light System with Multi-Head Attention Deep Q-Learning has shown that reinforcement learning with attention is a feasible and practical solution for adaptive traffic signal control. Our system not only decreases the average waiting time and queue of vehicles, but also always gives consistent priority to emergency vehicles, the most important security objective of this project. This project demonstrates that traffic lights don't have to be static and mechanical. Rather, with the proper use of artificial intelligence, they can be a smart system that can learn, change and evolve to benefit the community and the city.

7.2 Recommendation

While the objectives of the project have been achieved, there are some areas where the system could be enhanced and extended in future research. The following suggestions are made from the problems and limitations encountered in this project.

7.2.1 Multi-Intersection Network Scaling

The current system operates on a single isolated four-way intersection. Future work should extend the MHA-DQN framework to coordinate multiple intersections simultaneously, where agents must cooperate to prevent congestion from propagating across a network. Multi-agent reinforcement learning approaches, such as those using Graph Attention Networks, could be explored to model inter-intersection dependencies.

7.2.2 Proactive Emergency Vehicle Green Wave

The current emergency preemption responds reactively when a vehicle is within 150 metres. A future enhancement would implement a proactive green wave — coordinating multiple consecutive traffic signals to clear a corridor in advance of an approaching emergency vehicle, further reducing response time across longer routes

7.2.3 Real Traffic Data instead of Simulation

All experiments were conducted in SUMO with synthetic vehicle demand. Integrating real-world traffic data from camera feeds or loop detectors would validate the system under authentic traffic patterns including irregular demand, incidents, and pedestrian flows.

7.2.4 Emergency Vehicle Type Differentiation

The current system treats all emergency vehicles with the same priority level. Future work could implement tiered priority — giving ambulances carrying critical patients higher precedence than routine police patrols — using vehicle-type-aware reward shaping and phase preemption logic.

7.3 Summary of Objectives Achievement

	Objective	Status	Evidence
1	Construct an interactive simulation environment system	✅ Achieved	EnhancedTrafficEnvironment with SUMO + TraCI, 4 directions, multi vehicle types
2	Apply adaptive traffic management algorithms using real-time detector data	✅ Achieved	MHA-DQN selects phase + duration dynamically; queue reduced from ~50 to ~15 vehicles
3	Incorporate emergency vehicle recognition and prioritization	✅ Achieved	150m preemption override + mid-phase interrupt; RL emergency wait ≈ 0 vs baseline spikes up to 250s
4	Offer manual control features and graphical visualizations	✅ Achieved	SUMO-GUI, training plots every 10 episodes, model checkpoints every 20 episodes, JSON export
5	Compare MHA-DQN against simpler control approaches	✅ Achieved	Three-agent evaluation across S1, S2, S3; MHA-DQN reduces emergency wait by 99.4% vs Vanilla DQN (904.30s vs 5.60s in S3) and 85.9% vs Fixed-Time Baseline; avg waiting time 51–54s vs Vanilla 143–178s

Table 7.1: Summary of project objectives achievement.

REFERENCES

[1]Chandel, S., Yadav, S., & Yadav, S. (2018). Modern traffic control system. Malaya Journal of Matematik, 6(S1), 19–22. https://www.researchgate.net/publication/322752828_Modern_traffic_control_system

[2]Levy, J. I., Buonocore, J. J., & von Stackelberg, K. (2010). Evaluation of the public health impacts of traffic congestion: a health risk assessment. Environmental Health, 9(1). https://doi.org/10.1186/1476-069x-9-65

[3]Karmakar, G., Chowdhury, A., Kamruzzaman, J., & Gondal, I. (2020). A Smart Priority Based Traffic Control System for Emergency Vehicles. IEEE Sensors Journal, 1–1. https://doi.org/10.1109/jsen.2020.3023149

[4]Ghazal, B., ElKhatib, K., Chahine, K., & Kherfan, M. (2016). Smart traffic light control system. 2016 Third International Conference on Electrical, Electronics, Computer Engineering and Their Applications (EECEA). https://doi.org/10.1109/eecea.2016.7470780

[5]Diaz, N., Guerra, J., & Nicola, J. (2018). Smart Traffic Light Control System. 2018 IEEE Third Ecuador Technical Chapters Meeting (ETCM). https://doi.org/10.1109/etcm.2018.8580282

[6]N. M. Z. Hashim1, A. S. Jaafar2, N. A. Ali3, L. Salahuddin4, N. R. Mohamad5, M. A. Ibrahim6, “Traffic Light Control System for Emergency Vehicles Using Radio Frequency”, IOSR Journal of Engineering (IOSRJEN). https://www.academia.edu/download/106453165/3021-03754352.pdf

[7] Ms, F., & Sinhmar, P. (n.d.). INTELLIGENT TRAFFIC LIGHT AND DENSITY CONTROL USING IR SENSORS AND MICROCONTROLLER. In International Journal of Advanced Technology & Engineering Research. Retrieved April 14, 2025, from http://ijater.com/Files/b08bf563-bdc8-4048-974c936d2924d768_IJATER_03_06.pdf

[8]Orosz, G., Wilson, R. E., & Stépán, G. (2010). Traffic jams: dynamics and control. Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences, 368(1928), 4455–4479. https://doi.org/10.1098/rsta.2010.0205

[9]SIMULATION OF TRAFFIC SYSTEMS - AN OVERVIEW. (2025). Publish.uwo.ca. https://publish.uwo.ca/~jmalczew/gida_5/Pursula/Pursula.html#References

[10]Fagnant, D. J., & Kockelman, K. (2015). Preparing a nation for autonomous vehicles: opportunities, barriers and policy recommendations. Transportation Research Part A: Policy and Practice, 77, 167–181. https://doi.org/10.1016/j.tra.2015.04.003

[11]Hasan Haydar Yıldız, Furkan Güney, Ilhan Tunç, & Mehmet Turan Söylemez. (2024). Control of Emergency Vehicles with Deep Q-Learning. 1(1), 33–38. https://www.researchgate.net/publication/382447793_Control_of_Emergency_Vehicles_with_Deep_Q-Learning

[12]Park, S., Han, E., Park, S., Jeong, H., & Yun, I. (2021). Deep Q-network-based traffic signal control models. PLOS ONE, 16(9), e0256405. https://doi.org/10.1371/journal.pone.0256405

[13]Zeng, J., Hu, J., & Zhang, Y. (2018, June 1). Adaptive Traffic Signal Control with Deep Recurrent Q-learning. IEEE Xplore. https://doi.org/10.1109/IVS.2018.8500414

REFERENCES FOR IMAGE

Figure 1.1.2: Simulation-based Environment Traffic System: “Sharma, A. (n.d.). Traffic light management using AI [GitHub repository],” GitHub. https://github.com/adi-sharma707/traffic-light-management-using-AI

Figure 1.2.1: Stress Due to High Traffic: “Everypixel. (n.d.). [Image of traffic AI concept],” Everypixel. https://www.everypixel.com/image-5590480709813530123

Figure 1.2.2: Accident: “Woodyard, C. (2016, December 13),” Self-driving cars to start talking to each other. USA Today. https://www.usatoday.com/story/money/cars/2016/12/13/self-driving-cars-vehicle-to-vehicle-technology-v-to-v-nhtsa-autonomous-cars-v2v/95367310/

Figure 1.2.3: Emergency Vehicle Stuck in traffic: “SBS Hindi. (n.d.). Blocking an ambulance in India will now be fined Rs 10,000 under new law,” [Podcast episode]. SBS. https://www.sbs.com.au/language/hindi/en/podcast-episode/blocking-an-ambulance-in-india-will-now-be-fined-rs-10-000-under-new-law/rpiu90t9q

Figure 1.5.1: Traffic Congestion: “Bomey, N. (2015, July 29),” New car sales soaring, but cars getting older, too. USA Today https://www.usatoday.com/story/money/2015/07/29/new-car-sales-soaring-but-carsgetting-older-too/30821191/

Figure 1.5.2: Simulation-based Environment Traffic System: “Sharma, A. (n.d.). Traffic light management using AI [GitHub repository],” GitHub. https://github.com/adi-sharma707/traffic-light-management-using-AI

Figure 2.1.1: Ambulance Detection 1: “Srinivasan, K. (2020). Successful detection of an ambulance in traffic. In A. Raman, R. Kvs, & M. Moharir (Eds.), A hybrid framework for expediting emergency vehicle movement on Indian roads (Fig. 2),” ResearchGate. https://www.researchgate.net/figure/Successful-detection-of-anambulance-in-traffic_fig2_340893475

Figure 2.1.2: Ambulance Detection 2: “Figure 7. Maps of Laue reflections from crystal 1 (a,b) and crystal 2 (c,d) showing pressure-induced broadening,” Adapted from Zhang, Zhang, Zhang, and Zhang (2022), Mathematics, 12(9), 1514. https://doi.org/10.3390/mat12091514

Figure 2.1.3: Simulation Traffic Control 1: "Sun, G., Qi, R., Liu, Y., & Xu, F. (2024)," A dynamic traffic signal scheduling system based on improved greedy algorithm. PLOS ONE, 19(3), e0298417. https://doi.org/10.1371/journal.pone.0298417

Figure 2.1.4: Simulation Traffic Control 2: "Wiley Online Library. (n.d.). Figure 5b from article ATR6616702 [Image]," Wiley Online Library. https://onlinelibrary.wiley.com/cms/asset/ab6497c3-f6b2-4c64-be5ad40c963876ef/atr6616702-fig-0005b-m.jpg

Figure 2.1.5: Traffic light Detect Emergency Vehicle: Drive. (2023, July 25). Ford green-lights for emergency vehicles. Drive. https://www.drive.com.au/news/ford-green-lights-for-emergency-vehicles

Figure 2.1.6: Traffic light change based on situation: Ergün, S. (2023). Deep reinforcement learning at scramble intersections for traffic signal control: An example of Shibuya Crossing. In Science, Engineering Management and Information Technology (Vol. 1809, pp. 107–120). Springer. https://doi.org/10.1007/978-3-031-40398-9_7

Figure 2.2.1: Ambulance Simulation Environment: Yıldız, H., Güney, F., Tunc, I., &

Söylemez, M. T. (2024). Representation of actions. In Control of emergency vehicles with deep Q-learning. ResearchGate. https://www.researchgate.net/figure/Representation-of-Actions_fig4_382447793

Figure 2.2.2: Non prioritized fire truck in traffic: Alamy. (n.d.). Traffic jam motorway gridlock & fire engines on emergency services call to reach scene of accident as cars & vans try to move away on M25, UK. Alamy. Retrieved May 2, 2025, from https://www.alamy.com/traffic-jam-motorway-gridlock-fire-engines-onemergency-services-call-to-reach-scene-of-accident-as-cars-vans-try-to-move-away on-m25-uk-see-note-image364130027.html

Figure 2.2.3: Fire Truck in heavy traffic: Cuba, J. (2024, February 5). FDNY chief blames slower emergency response times on more cars. Streetsblog New York City. https://nyc.streetsblog.org/2024/02/05/fdny-chief-blames-slower-emergencyresponse-times-on-more-cars

Figure 2.2.4: High Cost: Wordwall.net. (n.d.). BTEC community resources. Wordwall. Retrieved May 2, 2025, https://wordwall.net/en-gb/community/btec

Figure 2.2.5: Panic Attack: CalmClinic. (n.d.). What to do if you experience a panic attack while driving. CalmClinic. https://www.calmclinic.com/panic/attacks/while-driving

Figure 2.3.1: Emergency Vehicle Prioritize: "SUMO Documentation. (n.d.). Emergency," Eclipse SUMO - Simulation of Urban MObility. Retrieved May 2, 2025. https://sumo.dlr.de/docs/Simulation/Emergency.html

Figure 2.3.2: Priority level: "Sarathambekai, S., Vairam, T., & Dharani, A. (2020)," Dynamic traffic light scheduling for emergency vehicles using fog computing. In Proceedings of the International Conference on Artificial Intelligence, Smart Grid and Smart City Applications (AISGSC 2019) (pp. 709–720). Springer. https://doi.org/10.1007/978-3-030-24051-6_65

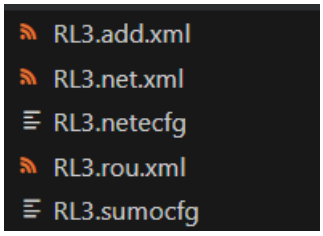
Figure 2.3.3: Traffic Signal detect Fire Truck: "Payson traffic signals ready for emergency vehicles," Payson Roundup. https://www.paysonroundup.com/news/local/payson-traffic-signals-ready-foremergency-vehicles/article_f42db256-5a54-566d-980f-7160f0f3500d.html

Figure 2.3.4: Vehicle Type: "Bhat, D., & Ghosh, P. (2019)", Examples of emergency car detection with our method [Image]. In Emergency vehicle detection. ResearchGate.https://www.researchgate.net/publication/332255419/figure/fig4/AS:845119590760449@1578503620159/Examples-of-emergency-car-detection-with-our-method-The-top image-shows-a-good-example.jpg

Figure 2.3.5: Observe real-time traffic control: "Roy, S., & Rahman, M. S. (2019)", Examples of emergency car detection with our method. In Emergency vehicle detection on heavy traffic road from CCTV footage using deep convolutional neural network (Fig. 4). IEEE.

APPENDIX

SUMO Configuration



RL3.add.xml

RL3.add.xml

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

<!-- generated on 2025-05-23 13:17:10 by Eclipse SUMO netedit Version 1.22.0
-->

<additional xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/additional_fi
    <!-- Detectors -->
    <laneAreaDetector id="EC_0" lane="EC_0" pos="115.00" length="115.00" period="300.00" file="e2_5.xml" friendlyPos="1"/>
    <laneAreaDetector id="EC_1" lane="EC_1" pos="115.00" length="115.00" period="300.00" file="e2_4.xml" friendlyPos="1"/>
    <laneAreaDetector id="NC_0" lane="NC_0" pos="90.00" length="113.00" period="300.00" file="e2_6.xml" friendlyPos="1"/>
    <laneAreaDetector id="NC_1" lane="NC_1" pos="90.00" length="113.00" period="300.00" file="e2_7.xml" friendlyPos="1"/>
    <laneAreaDetector id="SC_0" lane="SC_0" pos="40.00" length="115.00" period="300.00" file="e2_2.xml" friendlyPos="1"/>
    <laneAreaDetector id="SC_1" lane="SC_1" pos="40.00" length="115.00" period="300.00" file="e2_3.xml" friendlyPos="1"/>
    <laneAreaDetector id="WC_0" lane="WC_0" pos="80.00" length="115.00" period="300.00" file="e2_1.xml" friendlyPos="1"/>
    <laneAreaDetector id="WC_1" lane="WC_1" pos="80.00" length="115.00" period="300.00" file="e2_0.xml" friendlyPos="1"/>
</additional>

```

RL3.netecfg

RL3.netecfg

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

<!-- generated on 2025-05-23 13:17:11 by Eclipse SUMO netedit Version 1.22.0
-->

<neteditConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/net

    <input>
        <sumocfg-file value="RL3.sumocfg"/>
        <additional-files value="RL3.add.xml"/>
        <route-files value="RL3.rou.xml"/>
        <sumo-net-file value="RL3.net.xml"/>
    </input>

    <processing>
        <geometry.min-radius.fix.railways value="false"/>
        <geometry.max-grade.fix value="false"/>
        <offset.disable-normalization value="true"/>
        <lefthand value="1"/>
    </processing>

    <junctions>
        <no-turnarounds value="true"/>
        <junctions.corner-detail value="5"/>
        <junctions.limit-turn-speed value="5.50"/>
        <rectangular-lane-cut value="0"/>
    </junctions>

    <pedestrian>
        <walkingareas value="0"/>
    </pedestrian>

</neteditConfiguration>

```

RL3.rou.xml

RL3.rou.xml

```xml
<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">

    <vType id="normal_car" vClass="passenger" guiShape="passenger" color="0.7,0.7,0.7" maxSpeed="13.89" accel="2.6" decel="4.5" sigma="0.5" length="5.0" />
    <vType id="ambulance" vClass="emergency" guiShape="emergency" color="1,1,1" maxSpeed="22.0" accel="2.5" decel="4.5" length="6.5" />
    <vType id="police" vClass="emergency" guiShape="emergency" color="0,0,1" maxSpeed="25.0" accel="3.0" decel="5.0" length="5.0" />
    <vType id="firetruck" vClass="emergency" guiShape="emergency" color="1,0,0" maxSpeed="18.0" accel="2.0" decel="4.0" length="9.0" />



    <flow id="f_0" type="normal_car" begin="0" end="3600" from="WC" to="CS" vehsPerHour="160" />
    <flow id="f_1" type="normal_car" begin="0" end="3600" from="WC" to="CE" vehsPerHour="310" />
    <flow id="f_2" type="normal_car" begin="0" end="3600" from="WC" to="CN" vehsPerHour="180" />


    <flow id="f_3" type="normal_car" begin="0" end="3600" from="NC" to="CW" vehsPerHour="120" />
    <flow id="f_4" type="normal_car" begin="0" end="3600" from="NC" to="CE" vehsPerHour="160" />
    <flow id="f_5" type="normal_car" begin="0" end="3600" from="NC" to="CS" vehsPerHour="270" />


    <flow id="f_6" type="normal_car" begin="0" end="3600" from="EC" to="CN" vehsPerHour="160" />
    <flow id="f_7" type="normal_car" begin="0" end="3600" from="EC" to="CW" vehsPerHour="270" />
    <flow id="f_8" type="normal_car" begin="0" end="3600" from="EC" to="CS" vehsPerHour="150" />


    <flow id="f_9" type="normal_car" begin="0" end="3600" from="SC" to="CN" vehsPerHour="230" />
    <flow id="f_10" type="normal_car" begin="0" end="3600" from="SC" to="CW" vehsPerHour="120" />
    <flow id="f_11" type="normal_car" begin="0" end="3600" from="SC" to="CE" vehsPerHour="150" />


</routes>
```

Class EnhancedTrafficDQN

```python
class EnhancedTrafficDQN(nn.Module):
    def __init__(self, spatial_features: int = CONFIG['spatial_features'],
                 temporal_length: int = CONFIG['temporal_length'],
                 num_phases: int = CONFIG['num_phases'],
                 max_duration: int = CONFIG['max_green_duration'],
                 min_duration: int = CONFIG['min_green_duration'],
                 duration_step: int = CONFIG['duration_step']):
        super().__init__()
        self.spatial_features = spatial_features
        self.temporal_length = temporal_length
        self.num_phases = num_phases
        self.max_duration = max_duration
        self.min_duration = min_duration
        self.duration_step = duration_step
        self.num_nodes = CONFIG['graph_nodes']
        self.duration_steps = (max_duration - min_duration) // duration_step + 1
        self.total_actions = num_phases * self.duration_steps
        self.d_model = 64

        self.spatial_embedding = nn.Linear(spatial_features, self.d_model)
        self.spatial_attention = MultiHeadAttentionBlock(self.d_model, num_heads=8)
        self.temporal_embedding = nn.Linear(spatial_features, self.d_model)
        self.pos_encoding = PositionalEncoding(self.d_model)
        self.temporal_attention = MultiHeadAttentionBlock(self.d_model, num_heads=8)

        self.fusion = nn.Sequential(
            nn.Linear(self.d_model * (self.num_nodes + 1), 128),
            nn.ReLU(),
            nn.Dropout(0.2),
            nn.Linear(128, 64),
            nn.ReLU(),
            nn.Linear(64, self.total_actions)
        )
```

Reward Function

```
def calculate_action_reward(self, total_queue: float, final_queues: Dict[str, float],
                            detector_data: Dict[str, Dict[str, float]], current_phase: int) -> float:
    if detector_data is None:
        return 0.0

    EMERGENCY_TYPES = {"ambulance", "police", "firetruck"}

    queue_penalty = (total_queue / 100.0) * 0.5

    throughput_reward = 0.0
    for direction, data in detector_data.items():
        current_count = data['vehicle_count']
        throughput = max(0, self.vehicle_counts[direction] - current_count)
        throughput_reward += (throughput / 5.0) * 1.0
        self.episode_data['vehicles_passed_by_direction'][direction] += throughput
        self.vehicle_counts[direction] = current_count

    speed_reward = sum(min(data['mean_speed'] / 13.89, 1.0) * 0.3
                       for data in detector_data.values())

    current_halted = sum(1 for veh_id in traci.vehicle.getIDList()
                         if traci.vehicle.getSpeed(veh_id) < 0.1
                         and traci.vehicle.getTypeID(veh_id) not in EMERGENCY_TYPES)
    waiting_penalty = (current_halted / 10.0) * 0.2

    queue_balance = np.std([final_queues[d] for d in ['west', 'east', 'north', 'south']])
    balance_reward = -queue_balance * 0.5

    flow_reward = sum(data['mean_speed'] for data in detector_data.values()) / 4 * 0.2

    emergency_reward = 0.0
    for veh_id in traci.vehicle.getIDList():
        vtype = traci.vehicle.getTypeID(veh_id)
        if vtype not in EMERGENCY_TYPES:
            continue
        speed     = traci.vehicle.getSpeed(veh_id)
        wait_time = traci.vehicle.getWaitingTime(veh_id)
        next_tls  = traci.vehicle.getNextTLS(veh_id)
        if speed > 8.0:
            emergency_reward += 5.0
        if speed > 12.0 and not next_tls:
            emergency_reward += 3.0
        if speed < 0.5:
            penalty = min(wait_time / 5.0, 8.0)
            emergency_reward -= penalty
        if next_tls:
            tls_id, _, dist, _ = next_tls[0]
            if tls_id == self.junction_id and dist < 50.0:
                direction = get_incoming_direction(veh_id)
                desired_phase = EMERGENCY_DIRECTION_TO_PHASE.get(direction)
                if desired_phase is not None and current_phase != desired_phase:
                    emergency_reward -= 10.0

    phase_switch_penalty = 0.0
    if self.should_switch and current_phase == self.last_phase:
        phase_switch_penalty = -3.0

    reward = (
        np.clip(emergency_reward, -15.0, 15.0)
        + throughput_reward
        + speed_reward
        + flow_reward
        + balance_reward
        - queue_penalty
        - waiting_penalty
        + phase_switch_penalty
    )

    self.episode_data['total_queue_length'] += total_queue
    self.total_vehicles_passed = sum(self.episode_data['vehicles_passed_by_direction'].values())
    self.episode_data['vehicles_passed'] = self.total_vehicles_passed
    for direction, final_queue in final_queues.items():
        self.episode_data['queue_by_direction'][direction].append(final_queue)

    return reward
```

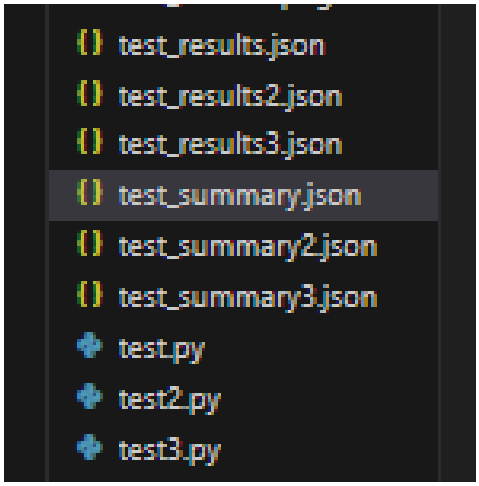
Emergency Override

```
def _get_emergency_phase(self) -> int:
    EMERGENCY_TYPES = {"ambulance", "police", "firetruck"}
    for veh in traci.vehicle.getIDList():
        if traci.vehicle.getTypeID(veh) not in EMERGENCY_TYPES:
            continue
        direction = get_incoming_direction(veh)
        if direction is None:
            continue
        desired_phase = EMERGENCY_DIRECTION_TO_PHASE.get(direction)
        if desired_phase is None:
            continue
        next_tls_list = traci.vehicle.getNextTLS(veh)
        if not next_tls_list:
            continue
        tls_id, _, dist, _ = next_tls_list[0]
        if tls_id == self.junction_id and dist <= EMERGENCY_PREEMPT_DISTANCE:
            logger.info(
                f"Emergency detected: {veh} ({traci.vehicle.getTypeID(veh)}) "
                f"at {dist:.1f}m → needs phase {desired_phase}"
            )
            return desired_phase
    return None
```

Replay Buffer

```python
class PrioritizedReplayBuffer:
    def __init__(self, capacity: int, alpha: float = 0.6, beta: float = 0.4):
        self.capacity = capacity
        self.alpha = alpha
        self.beta = beta
        self.buffer = []
        self.priorities = np.zeros(capacity, dtype=np.float32)
        self.pos = 0
        self.size = 0

    def push(self, state, action, reward, next_state, done):
        max_priority = self.priorities.max() if self.size > 0 else 1.0
        experience = Experience(state, action, reward, next_state, done, max_priority)
        if self.size < self.capacity:
            self.buffer.append(experience)
            self.priorities[self.pos] = max_priority
            self.size += 1
        else:
            self.buffer[self.pos] = experience
            self.priorities[self.pos] = max_priority
        self.pos = (self.pos + 1) % self.capacity

    def sample(self, batch_size):
        priorities = self.priorities[:self.size]
        probs = priorities ** self.alpha
        probs /= probs.sum()
        indices = np.random.choice(self.size, batch_size, p=probs)
        samples = [self.buffer[idx] for idx in indices]
        weights = (self.size * probs[indices]) ** (-self.beta)
        weights /= weights.max()
        return samples, indices, weights

    def update_priorities(self, indices, priorities):
        for idx, priority in zip(indices, priorities):
            self.priorities[idx] = priority + 1e-5

    def __len__(self):
        return self.size
```

Testing Files



Testing 1

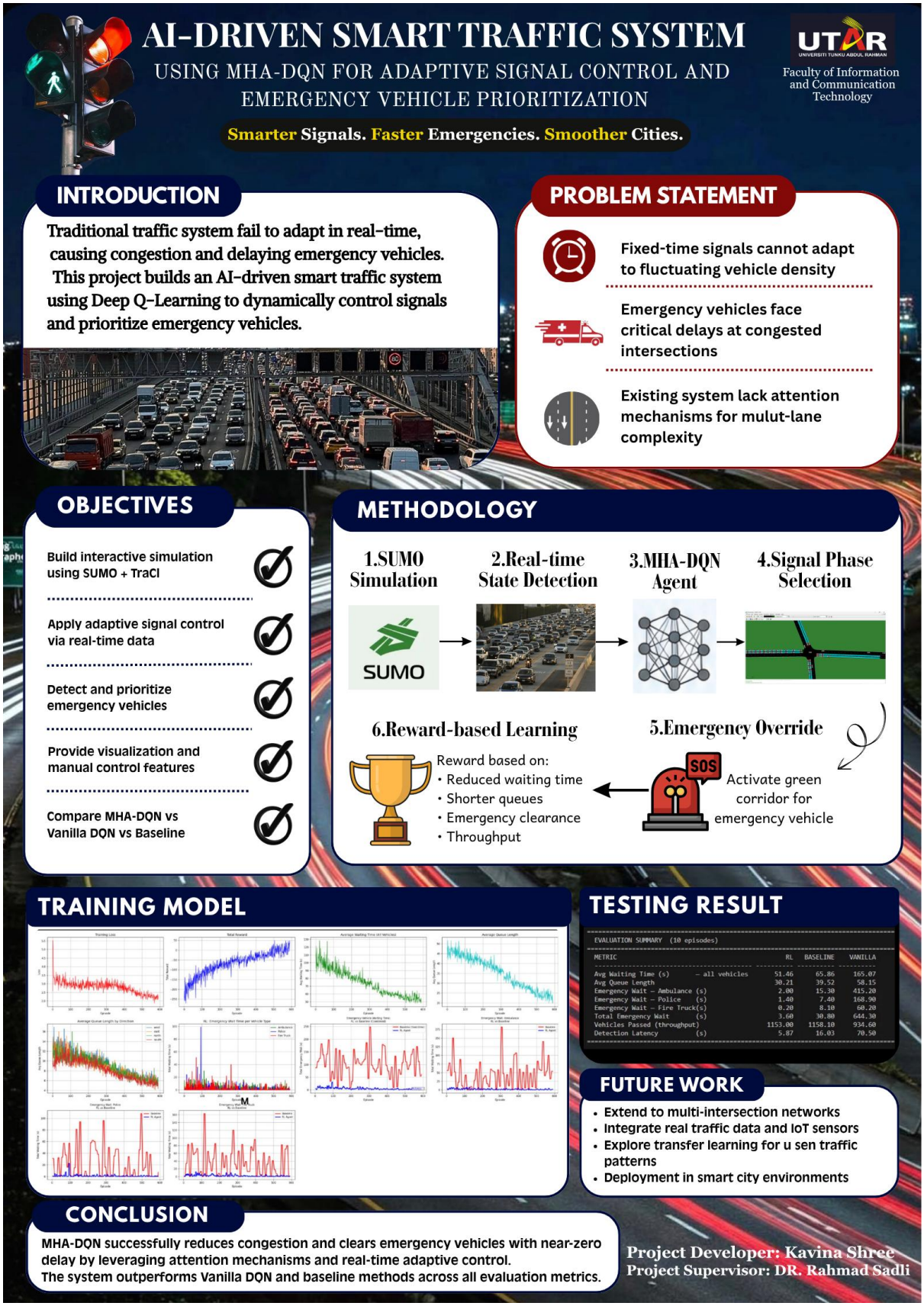
Testing 2

test_results2.json > ...


```
{
  "eval_config": {
    "num_test_episodes": 10,
    "rl_model_path": "traffic_model.pth",
    "vanilla_model_path": "vanilla_model.pth",
    "emergency_vehicles_per_episode": 3,
    "sumo_config_file": "RL3.sumocfg"
  },
  "rl": [
    {
      "avg_waiting_time": 52.365284974093264,
      "avg_queue_length": 30.917441860465118,
      "emerg_wait_total": 13.0,
      "emerg_wait_ambulance": 13.0,
      "emerg_wait_police": 0.0,
      "emerg_wait_firetruck": 0.0,
      "vehicles_passed": 1158,
      "detect_latency_mean": 2.3333333333333335,
      "detect_latencies": [
        0.0,
        0.0,
        7.0
      ]
    },
    {
      "avg_waiting_time": 52.53543307086614,
      "avg_queue_length": 30.974690994702765,
      "emerg_wait_total": 0.0,
      "emerg_wait_ambulance": 0.0,
      "emerg_wait_police": 0.0,
      "emerg_wait_firetruck": 0.0,
      "vehicles_passed": 1143,
      "detect_latency_mean": 3.6666666666666665,
      "detect_latencies": [
        7.0,
        0.0,
        4.0
      ]
    },
    {
      "avg_waiting_time": 52.12543252595156,
      "avg_queue_length": 30.64819136522754,
      "emerg_wait_total": 0.0,
      "emerg_wait_ambulance": 0.0,
      "emerg_wait_police": 0.0,
      "emerg_wait_firetruck": 0.0,
      "vehicles_passed": 1156,
      "detect_latency_mean": 8.333333333333334,
      "detect_latencies": [
        6.0,
        7.0,
        12.0
      ]
    },
    {
      "avg_waiting_time": 49.88927637314734,
      "avg_queue_length": 28.98411764705824,
      "emerg_wait_total": 8.0,
      "emerg_wait_ambulance": 0.0,
      "emerg_wait_police": 8.0,
      "emerg_wait_firetruck": 0.0,
      "vehicles_passed": 1147,
      "detect_latency_mean": 5.333333333333333,
      "detect_latencies": [
        8.0,
        0.0,
        8.0
```

Testing 3

Poster


AI-DRIVEN SMART TRAFFIC SYSTEM
USING MHA-DQN FOR ADAPTIVE SIGNAL CONTROL AND EMERGENCY VEHICLE PRIORITIZATION
Smarter Signals. Faster Emergencies. Smoother Cities.
UTAR
UNIVERSITI TUNKU ABDUL RAHMAN
Faculty of Information and Communication Technology
INTRODUCTION
Traditional traffic system fail to adapt in real-time, causing congestion and delaying emergency vehicles. This project builds an AI-driven smart traffic system using Deep Q-Learning to dynamically control signals and prioritize emergency vehicles.
PROBLEM STATEMENT
Fixed-time signals cannot adapt to fluctuating vehicle density
Emergency vehicles face critical delays at congested intersections
Existing system lack attention mechanisms for mulut-lane complexity
OBJECTIVES
Build interactive simulation using SUMO + TraCI
Apply adaptive signal control via real-time data
Detect and prioritize emergency vehicles
Provide visualization and manual control features
Compare MHA-DQN vs Vanilla DQN vs Baseline
METHODOLOGY
1.SUMO Simulation
2.Real-time State Detection
3.MHA-DQN Agent
4.Signal Phase Selection
SUMO
6.Reward-based Learning
Reward based on:
• Reduced waiting time
• Shorter queues
• Emergency clearance
• Throughput
5.Emergency Override
SOS
Activate green corridor for emergency vehicle
TRAINING MODEL
TESTING RESULT
EVALUATION SUMMARY (10 episodes)
METRIC RL BASELINE VANILLA
Avg Waiting Time (s) – all vehicles 51.46 65.86 165.07
Avg Queue Length 30.21 39.52 58.15
Emergency Wait – Ambulance (s) 2.00 15.30 415.20
Emergency Wait – Police (s) 1.40 7.40 168.90
Emergency Wait – Fire Truck(s) 0.20 8.10 60.20
Total Emergency Wait (s) 3.60 30.80 644.30
Vehicles Passed (throughput) 1153.00 1158.10 934.60
Detection Latency (s) 5.87 16.03 70.50
FUTURE WORK
• Extend to multi-intersection networks
• Integrate real traffic data and IoT sensors
• Explore transfer learning for u sen traffic patterns
• Deployment in smart city environments
CONCLUSION
MHA-DQN successfully reduces congestion and clears emergency vehicles with near-zero delay by leveraging attention mechanisms and real-time adaptive control.
The system outperforms Vanilla DQN and baseline methods across all evaluation metrics.
Project Developer: Kavina Shree
Project Supervisor: DR. Rahmad Sadli