AI-Driven Adaptive Traffic Light Optimization Using Dueling Deep Q-Networks

By

Lau Yee Hang

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 report 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 report 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 to my supervisor, **Dr. Rahmad Sadli**, for his expert guidance, constructive feedback and continuous support throughout this project on adaptive traffic signal control using reinforcement learning. His advice and encouragement were invaluable in shaping the work and helping me complete this research.

My thanks also go to the UTAR lecturers and the project moderator for their teaching, helpful discussions and administrative support during the course. Their insights and suggestions improved both the technical quality and presentation of this study.

I am deeply grateful to my parents for their unwavering love, patience and encouragement throughout my studies. Finally, I would like to thank my friends for their moral support and for being there when I needed motivation during the project.

ABSTRACT

This study presents a Dueling Double Deep Q-Network (Dueling DDQN) based adaptive traffic signal control framework for a signalised four-way intersection simulated in SUMO (Simulation of Urban Mobility). Unlike traditional fixed-time controllers, the proposed agent dynamically selects among three discrete signal phases based on a 13-dimensional state vector encoding one-hot phase representation, directional vehicle queues and waiting times, and pedestrian demand. The reward function is a sevencomponent composite that balances vehicle efficiency through queue reduction and throughput bonuses, safety through a flat red-light violation penalty, and pedestrian service through a fixed penalty weight.

A controlled ablation study compares four DQN architectures — VanillaDQN, DoubleDQN, DuelingDQN, and DuelingDoubleDQN — under identical reward functions, hyperparameters, and training episodes across three demand scenarios (low: ~905 veh/hr, medium: ~1,809 veh/hr, high: ~3,165 veh/hr). All models are trained on the medium scenario only, and generalisation is evaluated by testing on unseen low and high demand scenarios without retraining. Results demonstrate that DuelingDoubleDQN achieves a 54.1% reduction in average vehicle waiting time and a 49.3% reduction in vehicle queue length on the training scenario compared to VanillaDQN. Under the unseen high-demand scenario, DuelingDoubleDQN reduces vehicle wait by 21.5%, confirming policy generalisation beyond the training distribution. A key ablation finding is that DuelingDQN without the double correction performs worst at high demand (1,095.98s wait), exceeding even VanillaDQN, which confirms that the double update rule is critical for stable performance under near-saturation conditions. An automated red-light violation monitoring module detects, classifies, and logs each violation event with timestamp, vehicle ID, approach direction, speed, and phase state, enabling precision/recall/F1 evaluation. A reward design limitation regarding pedestrian service deprioritisation is identified and investigated through a constrained reward variant. Findings contribute towards safer, more adaptive, and computationally tractable intelligent transportation systems.

Area of Study: Intelligent Transportation Systems; Artificial Intelligence, Deep Learning

Keywords: Adaptive Traffic Signal Control, Dueling Double Deep Q-Network, Deep Reinforcement Learning, Traffic Simulation, Violation Detection.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

Q(s,a)	action-value function / Q-value
V(s)	state-value (value stream in Dueling architecture)
A(s,a)	advantage function (advantage stream)
a_t	action at time t
$\varphi 0, \varphi 1, \varphi 2$	one-hot encoding of the current signal phase
q_D	queue length on approach D ($D \in \{E, N, W, S\}$)
w_D	total waiting time on approach D
ε (epsilon)	exploration rate in ε-greedy policy; decay schedule described
γ (gamma)	discount factor
W_veh	Total vehicle waiting time (seconds)
W_ped	Total pedestrian waiting time (seconds)
Q_t	Total vehicle queue at step t
ped_count	Normalised pedestrian queue count
ped_wait	Normalised pedestrian waiting time
τ	Soft update rate = 0.005
α	Learning rate = 1×10^{-4}
TP / FP / FN	True Positive / False Positive / False Negative

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ATCS	Adaptive Traffic Control System
API	Application Programming Interface
CSV	Comma-Separated Values
CUDA	Compute Unified Device Architecture
DDPG	Deep Deterministic Policy Gradient
DQN	Deep Q-Network
DDQN	Double Deep Q-Network
Dueling DQN	Dueling Deep Q-Network
Dueling DDQN	Dueling + Double DQN combination
Env	Environment
DRL	Deep Reinforcement Learning
EC / NC / WC / SC	East / North / West / South → Centre
F1	F1 Score
FYP	Final Year Project
GPU	Graphics Processing Unit
GAT	Grounded Action Transformation
IDE	Integrated Development Environment
ITS	Intelligent Transportation System
LLM	Large Language Model
MARL	Multi-Agent Reinforcement Learning
MSE	Mean Squared Error (loss)
MDP	Markov Decision Process
PPO	Proximal Policy Optimization
PER	Prioritised Experience Replay
RL	Reinforcement Learning
ReLU	Rectified Linear Unit
SCATS	Sydney Coordinated Adaptive Traffic System
SCOOT	Split Cycle Offset Optimisation Technique

SUMO	Simulation of Urban Mobility
TraCI	Traffic Control Interface
UGAT	Uncertainty-aware Grounded Action Transformation
TD / TD-error	Temporal-Difference / TD error
TP / FP / FN	True Positive / False Positive / False Negative
Veh/h	vehicles per hour
V/A	Value / Advantage stream notation

CHAPTER 1 - Introduction

This chapter introduces the motivation and background of the project, the main problems of current traffic signal control systems, and the key research objectives. It also formulates the project scope and proposed contributions of the project in applying AI-based methods, in particular Dueling Deep Q-Networks, to improve traffic light optimization and aid traffic law enforcement for urban mobility.

1.1 Problem Statement and Motivation

Urban road networks of the contemporary era are confronted with substantial traffic jams. This problem is widespread in cities globally and it does not have an impact on the size and the economic position of the city. The reasons for such a state are numerous: the demographics in urban areas indicate the growth of the population, private car ownership is on the rise and road networks are usually not designed to cope with all those vehicles moving on them. Of course, the picture is not brighter when we see that transportation is required more than the available infrastructure. Traffic congestion is tracking back to unfriendly signal patterns and lack of inter-signal coordination the most.

Notably, the usual case is that traffic signals are time-fixed and do not change according to the real-time traffic. For example, traffic signals are still run at many intersections on pre-timed patterns which are green, amber or red according to the old traffic surveys. One disadvantage of these fixed-time systems is that they cannot react to sudden changes in the traffic flow such as an unexpected number of cars that arrive at the minor road. Consequently, these green phases can be present on empty lanes at the same time with others that are waiting at the red light, thus, the capacity of the road is being wasted while delays are deepened.

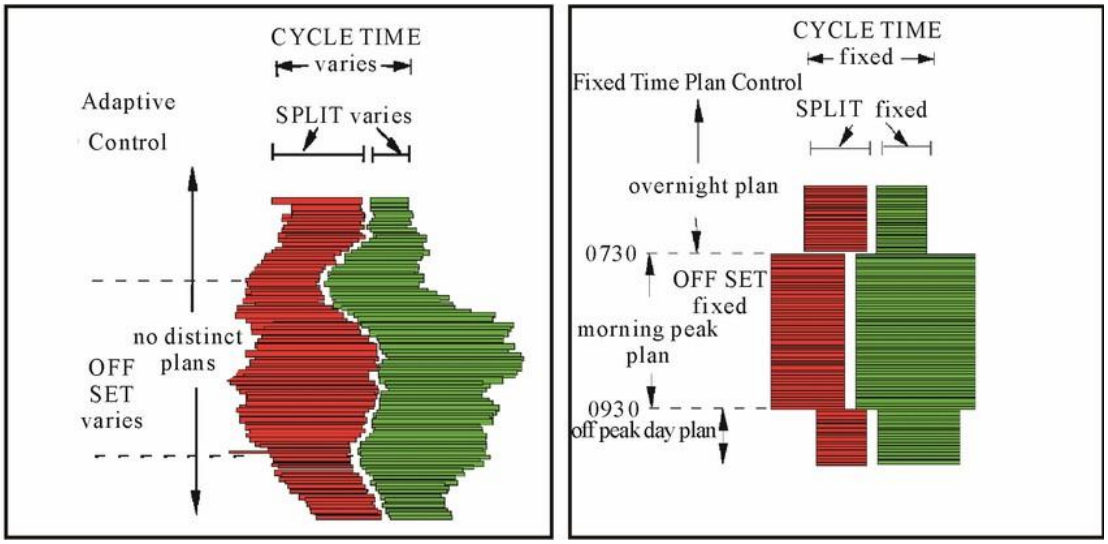


Figure 1.1 Comparison between traditional fixed-time and intelligent adaptive traffic signal control systems.[20]

As shown in Figure 1.1, adaptive systems can dynamically adjust signal phases to match real-time traffic demand, overcoming the inefficiencies of fixed-time control. This problem is not unique to any one country. Urban centres such as Kuala Lumpur, Los Angeles, London, and Jakarta routinely suffer severe peak-hour delays due to inefficient traffic signals.

Delays of traditional signal control can be categorized as several interrelated issues:

- **Rigid and inefficient control systems.**

  Fixed-time control presumes stationarity in traffic demand and relies on pre-computed timing plans. Even simple vehicle-actuated systems, which extend, or truncate green intervals based on local detections, are rule-based and usually operate on short-term thresholds. These methods lack the flexibility to allocate green time opportunistically across approaches when flow patterns change rapidly (e.g., due to incidents, special events, or time-varying demand). The result is wasted green time on under-utilised approaches and persistent queues on oversaturated approaches.
- **Lack of real-time adaptive learning and anticipation.**

  Existing controllers are predominantly reactive: they respond to present sensor states but generally do not learn from historical trends or anticipate imminent traffic developments. The absence of learning-on-the-go prevents controllers from exploiting temporal regularities (rush-hour patterns, turning movement trends, or recurring

platoons) and from taking proactive measures that could prevent congestion propagation. In dense, multimodal environments where vehicle types, weather, and human behaviour interact in complex ways, purely reactive strategies are insufficient.

- **Neglect of network-wide and coordination effects.**
  Traditional optimisation often focusses on individual intersections, tuning timings to minimise delay locally without considering downstream or upstream consequences. However, urban traffic operates as a network; changes at one junction can create spillback or starvation at neighbouring junctions. Without coordination or network-aware strategies, local improvements may produce adverse systemic effects.
- **Insufficient integration of safety and multi-modal considerations.**
  Many performance studies prioritise efficiency metrics (delay, queue length, throughput) while paying less attention to safety-related outcomes (e.g., red-light violations) and pedestrian experience. A controller that improves throughput but increases unsafe behaviours or imposes excessive pedestrian waiting undermines the overall objective of safe, equitable urban mobility.

These shortfalls motivate the employment of adaptive, experience-learning control strategies that can (i) sense and adapt to the current traffic state, (ii) learn through interaction and experience and make improved decisions over time, and (iii) be studied both for efficiency and safety analysis purposes. Deep Reinforcement Learning (DRL) offers an intuitive formalism: a traffic signal controller can be conceptualized as an agent interacting with a traffic environment and learning policies that maximize the best long-term performance. Value-based DRL models such as Deep Q-Networks are widely used for discrete-action problems such as phase selection because they provide an intuitively interpretable mechanism for learning action-value functions from high-dimensional inputs (e.g., lane occupancies, queue lengths and waiting times).

More sophisticated architectures, such as Dueling Deep Q-Networks, address specific problems of learning experienced in regulating traffic flow. The Dueling architecture separates state-value and action-advantage streams, and therefore the net can better approximate the value of states irrespective of specific actions; along with Double Q-learning, it reduces overestimation of the Q-values. These features can help enable improved learning stability and policy robustness within noisy, sparse-reward traffic conditions where unstable value estimates can cause

oscillatory or hazardous signalling behaviour.

Because of safety and practical concerns, simulation-first is the method used: the extensible, reproducible environment for developing road networks, flows and instrumentation (loops, lane-area detectors) and detailed safety and performance logging is provided by the open-source SUMO platform. Simulation enables systematic testing under multiple demand situations, vehicle mixes and control policies without risk of harm to road users.

Briefly, the motivation for the work is to advance intersection control beyond hard-coded, reactive schemes toward adaptive, learning controllers evaluated holistically. Embracing algorithms for connecting stability and state/value decomposition (e.g., variants of Dueling DQN), and taking advantage of reproducible SUMO experiments, the work is designed to demonstrate measurable performance gains at the intersection scale and monitor safety and multi-modal service levels as a byproduct.

1.2 Research Objectives

Expanding the Problem Statement and Motivation (Section 1.1), this endeavour seeks to achieve three specific, quantifiable goals that are intended to enhance the operation, safety, and multi-modal service of the intersection by employing the method of reinforcement learning within a SUMO traffic simulation environment.

Objective 1 — Use Duelling Deep Q-Networks to design and develop an AI-Driven adaptive traffic signal control system.

Develop a Dueling DQN agent that selects discrete signal phases for an intersection in SUMO. The controller will learn from state observations (e.g., lane occupancies, queue lengths, waiting times) and be trained to maximise long-term performance. The system's effectiveness will be evaluated by generating and analysing standard performance metrics such as mean vehicle delay, total queue length, and intersection throughput under a range of demand scenarios.

Objective 2 — Simulate an automated monitoring system for detecting and logging traffic violations (red-light running).

Implement a SUMO-based detection module that identifies and records red-light violations and

abnormal approach behaviours. Each logged event will include metadata (timestamp, approach/direction, vehicle id or class, and violation type) and to evaluate the detector's accuracy using precision/recall/F1 metrics.

Objective 3 — Integrate pedestrian dynamics and adapt the Dueling DQN learning objective to balance vehicular efficiency and pedestrian waiting time.

To integrate the pedestrian dynamics within the SUMO scenarios and tune the Dueling DQN reward for a trade-off between motorised traffic efficiency and walking level-of-service. Evaluate the system-wide performance under multi-modal performance metrics (vehicle delay, walking times, queue lengths, throughput, and violation frequency) to reflect the way the controller strikes a balance between motorised traffic performance and walking level-of-service.

1.3 Project Scope and Direction

1.3.1 Scope

- **Primary objective:** design, implement and evaluate a Dueling DQN–based adaptive traffic signal controller for a standard four-way signalised intersection.
- **Simulation platform:** SUMO for creating traffic, sensors (lane-area/loop detectors), simulation of pedestrians and event logging.
- **Safety monitoring:** SUMO/TraCI module for red-light running and stop-line abnormal behaviour detection and logging (log messages include timestamp, vehicle id, approach and violation type).
- **Multi-modal modelling:** cars (multi-class) and pedestrian crossing demand were included; the pedestrian waiting is modelled through state and reward.
- **State & action:** unified, normalised state vector exposed to the agent (phase encoding + normalised queue lengths + normalised waiting times + number of pedestrians). Discrete action space and semantics (e.g., phase selection as a discrete action (one of three signal phases)) will be chosen, tracked and held constant across experiments.
- **Reward:** single composite reward used throughout experiments balancing vehicle efficiency, pedestrian wait time and safety penalties (parameterised for sensitivity analysis).

1.3.2 Project direction

1. **Environment standardisation**
    - Save and finish the state encoding, normalisation range, action encoding and reward definition. Employ the same interface for all ablation and experiments.
2. **Controller development**
    - Utilise the Dueling DQN agent (value and advantage streams) with experience replay, target network and normal stabilisation methods (reward scaling, gradient clipping, seed control).
3. **Violation & pedestrian modules**
    - Incorporate red-light running detection logging and pedestrian flow generators into SUMO scenarios.
4. **Experimental scenarios**
    - Three demand scenarios are implemented to evaluate policy generalisation. The low-demand scenario uses approximately 905 vehicles per hour and 900 pedestrians per hour (roughly 50% of the baseline). The medium-demand scenario uses approximately 1,809 vehicles per hour and 1,800 pedestrians per hour (the training scenario). The high-demand scenario uses approximately 3,165 vehicles per hour and 3,150 pedestrians per hour (approximately 175% of the baseline). All three scenarios share the same road network and detector configuration, differing only in the vehicle and pedestrian flow rates defined in separate route files (TrafficLight_rou_low.xml, TrafficLight_rou.xml, TrafficLight_rou_high.xml). Non-standard scenarios such as incident conditions and high-turn-volume situations are identified as a direction for future work in Chapter 7.
5. **Training & evaluation protocol**
    - Train agent under a known hyperparameter configuration, save interim checkpoints, and All scenarios use a fixed SUMO random seed, producing deterministic results. Each model is evaluated for 5 test episodes per scenario with epsilon = 0.
6. **Analysis & ablation**
    - Evaluate on metrics (below), compared against three DQN ablation baselines — VanillaDQN, DoubleDQN, and DuelingDQN — under identical conditions. The ablation isolates the contribution of the dueling architecture and the double

update rule independently for behaviour clarification purposes.

1.4 Contributions

This project contributes to the literature on intelligent control of traffic signals in the following principal ways:

First, it develops an adaptive control unit based on the architecture for the Dueling Deep Q-Network. The separation of the value and advantage estimates facilitates more stable and efficient learning for the framework, and that is why the dynamic signal phase can be adjusted as the response of the framework to the changing real-time traffic environment. That subsequently leads to the overall efficiency being improved through the minimization of the delay, the reduction of the lengths of the queues, and the increase of the throughput.

Secondly, the innovation involves incorporating pedestrian behaviours under the control framework. Most of the available methods opt for only optimising vehicle flows. Nevertheless, the approach takes into consideration walking times of the pedestrians and accounts for it while performing the learning process. Owing to cooperation by vehicular flows and pedestrian services, the controller offers more fairness and safety of intersection control.

Thirdly, the program implements a simulation surveillance system capable of recording and tracking unlawful traffic behaviour with the emphasis on red-light running. It can detect the offenders who run the red light or illegally stop their vehicles within the SUMO simulation, and therefore, a copy of the infringement as the time, place, and category can be captured. This is a clear manifestation of the degree to which intelligent traffic control systems and automatic intervention devices can render the road setting more secure.

Fourth, the project establishes a SUMO-based simulation framework as a safe, flexible, and reproducible environment for experimentation. A realistic four-way intersection is modelled under diverse scenarios—including peak, off-peak, and sudden demand surges—allowing systematic evaluation of controller performance across a variety of traffic conditions.

Finally, the project conducts a comprehensive performance evaluation by comparing the

DuelingDoubleDQN controller against three DQN ablation baselines — VanillaDQN, DoubleDQN, and DuelingDQN. Key metrics such as vehicle delay, queue length, throughput, and frequency of violations are collected and analysed. The results demonstrate not only the efficiency gains but also the broader safety and multimodal benefits of the proposed approach.

Through these contributions, the project establishes Dueling DQN as a strong foundation for next-generation adaptive traffic control systems, offering improvements in efficiency, safety, and multimodal service that align with the goals of future Intelligent Transportation Systems (ITS).

1.5 Report Organization

The report is organised into seven chapters. Chapter 1 introduces the motivation, objectives, scope, and contributions. Chapter 2 reviews the literature on traffic signal control and deep reinforcement learning. Chapter 3 details the proposed methodology including the Dueling DDQN architecture, equations, and system design. Chapter 4 describes the full system design covering SUMO configuration, network architecture, reward function, violation monitoring module, and ablation study design. Chapter 5 documents the experimental setup, training procedure, and evaluation methodology. Chapter 6 presents and discusses the multi-scenario evaluation results including generalisation analysis and ablation findings. Chapter 7 concludes the study and provides recommendations for future work.

CHAPTER 2 - Literature Reviews

2.1 Previous works on Deep Learning

2.1.1 Evolution of Traffic Signal Control Systems

The management of the signalized junctions has been experienced a series of technical and methodological changes over previous years which has showing the increasing complexity of urban traffic load. Initially the systems were controlled by fixed-time traffic signal plans that cause the cycle duration, phase splits, and differences on time were predetermined from past traffic counts and manually programmed into controllers [1]. These systems were basis, low-cost to implement, and consistent in functioning. However, they exhibited fundamental weaknesses: they assumed stable and recurring traffic patterns, were unable to adapt to short-term fluctuations, and often resulted in wasted green time on low demand approaches while congested movements remained undersupplied [1].

Responsiveness was improved when actuated control was introduced in a subsequent generation. Actuated control also made use of vehicle detectors such as inductive loop sensors, magnetometers, and video-based detectors. These vehicle detectors were established at approaches to the intersection and provided immediate information on the presence of traffic. The green phase of a signal would then be extended or terminated by the controller based on the real time presence of traffic, this was typically accomplished through gap-out or extension logic [2]. The ability for the controller to adapt to present circumstances locally, eliminated unnecessary delays that existed with fixed-time operation. Despite the advantages, actuated control was still reactive and localised, the system merely optimised control on a single intersection basis. This meant that it could not anticipate any upstream platoons, nor could it facilitate a coordinated approach across multiple intersections along a corridor [2].

The limitations of purely localised strategies gave rise to adaptive traffic control systems (ATCS) in the 1980s and 1990s. Prominent examples include SCOOT and SCATS which introduced network-level optimisation through centralised or distributed architectures [3][4]. These systems use aggregated real-time traffic data (e.g., from detectors across the network) to adjust parameters such as cycle length, green splits, and offsets dynamically. SCOOT employs

a centralised optimisation process based on online traffic models, while SCATS adopts a decentralised approach in which each intersection adjusts its own splits and cycle length with some coordination across the network [3]. Both systems significantly outperform fixed-time and actuated controllers under variable demand, particularly in corridors with recurrent peak flows.

While they have delivered operational improvements, adaptive commercial systems have limitations. They require hours of manual calibration, utilize conventional traffic modelling tools for simplified traffic flows, and must be calibrated to local infrastructure and policy. In addition, given that the optimization logic for adaptive systems is rule-based and prescriptive, there is limited flexibility to account for unexpected conditions like accidents, non-recurrent congestion, and mixed-mode interactions (pedestrians, cyclist, or public transport priority). Furthermore, a maintenance cost of adaptive systems and the lack of specialized expertise to implement such systems. This lack of expertise is a large barrier to developing cities implementing these systems [4].

Figure 2.1 shows the general architecture of signal control systems and a progression from fixed time → actuated → adaptive. This sequential evolution indicates two continual gaps: (1) the demand for controllers that learn and adapt to stochastic and dynamic demand; and (2) the demand for methods that can consider local performance, at the intersections, vs at the network level (coordination). These gaps will motivate the discussion of artificial intelligence and reinforcement learning as a next-generation data analytics approach to signal control [1].

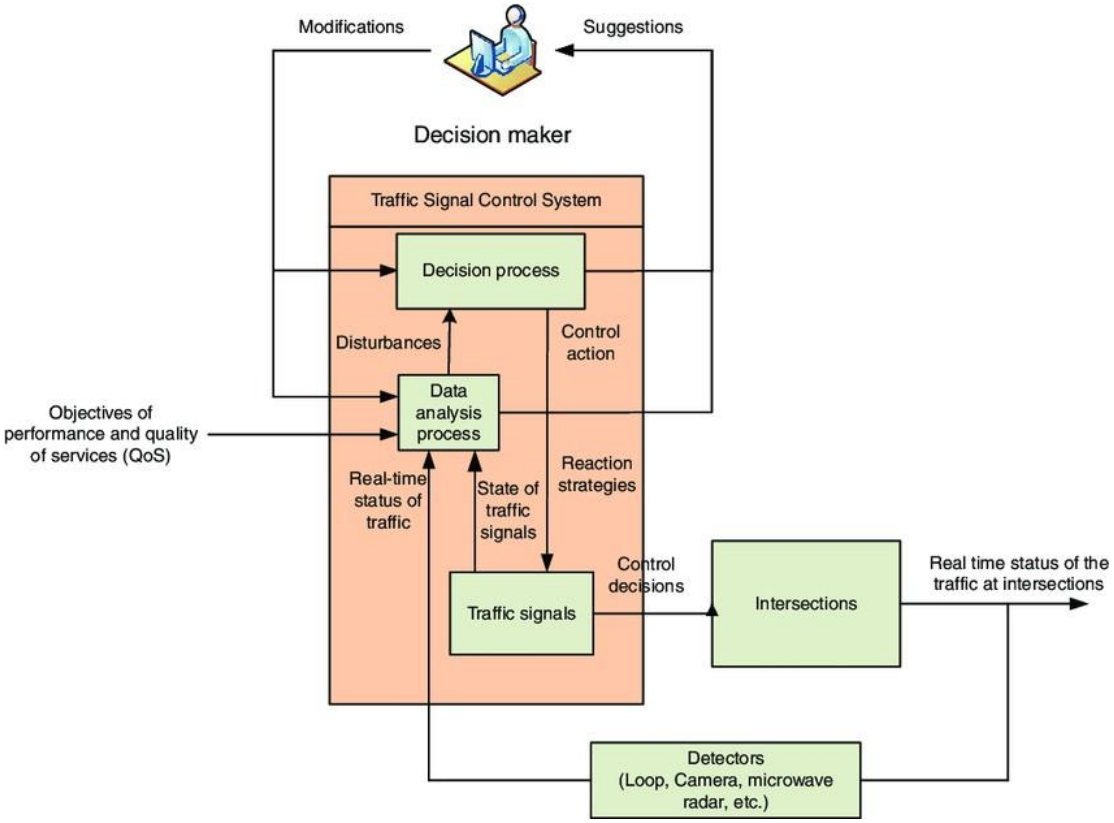


Figure 2.1 Traffic signal control system architecture. [21]

2.1.2 Emergence of DL and RL in Traffic Signal Control

The growing variability and uncertainty in urban traffic flows brought into relief the limitations of traditional adaptive systems and resulted in a need for more data-driven and learning-based methods. In this regard, deep learning and reinforcement learning (RL) & are now at the forefront of this evolution because these methods can process complex high-dimensional datasets and learn policies through interactions with the environment.

The earliest applications to deep learning and traffic management were largely focused on traffic flow prediction and signal timing optimisation also in supervised learning contexts, using either unsupervised learning or supervised learning. Neural networks were trained to predict vehicle counts, travel times, or congestion levels and then, these measures could be used to adjust signal plans [1][3]. While these methods demonstrably had improved modelling accuracy compared to classical statistical models, they still

were not able to adapt online and do not have the ability to update the learning model assuming that traffic conditions could be rapidly changing. Such rigidity meant that this method could not succeed in real-time adaptive signal control where continuous feedback is not only important, but daily adaptations scores are crucial and therefore speed of adaptation matters.

To address this gap, reinforcement learning (RL) emerged as a more natural perspective to represent traffic signal control. For RL, we model the traffic signal as an agent interacting with the traffic environment shown in Figure 2.2. The agent receives the state of the environment (e.g., queue lengths, lane occupancies, phase status), chooses an action (e.g., which phase to actuate or when to change phases), and incurs a reward which reflects the performance of the traffic (e.g., lower delays or shorter queue lengths.). Over multiple interactions the agent learns a policy that maximizes the cumulative reward, thereby optimizing signal timings over repeated trial-and-error.

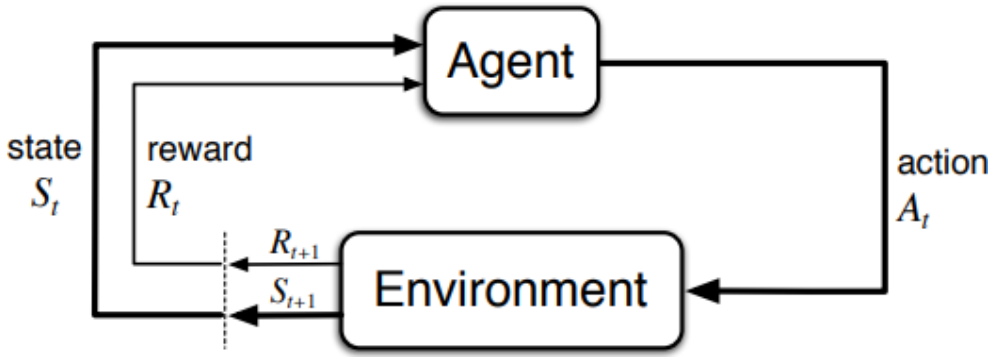


Figure 2.2 The agent–environment interaction in a Markov decision process.[22]

Conventional RL algorithms, such as Q-learning and SARSA, provided the conceptual underpinnings for value-based control. These algorithms were based on the estimate of the action-value function Q(s,a), utilizing temporal-difference updates in a bootstrapping fashion to learn and improve control policies [22]. Unfortunately, when it comes to real-world intersections, these tabular methods rarely scale as the state space becomes exceedingly large, considering the increase in variables by having multiple approach lanes with possible vehicle turning ratios, accounting for pedestrian traffic, and taking into consideration time-of-day variations. The consequences of this "curse of dimensionality" makes classical RL increasingly untenable for many traffic signal control applications.

Most progress on this topic came from advances in Deep Q-Networks (DQN). Both the Q-function and the value iterations which converge to optimal behaviours were being approximated using deep neural networks instead of the traditional tables to learn optimal strategies. DQN works by applying a neural network to map all input states derived from traffic features (e.g., queue lengths, vehicle positions) to Q-values denoting each possible action as shown in Figure 2.3 [29]. DQN implements two primary innovations to stabilise the training process; (1) experience replay that permits all past transitions to be stored in a memory buffer, which allows different mini-batches to be randomly sampled to reduce the correlation of successive training updates, and (2) a target network that is slowly updated to provide a fixed reference for Q-value estimates.

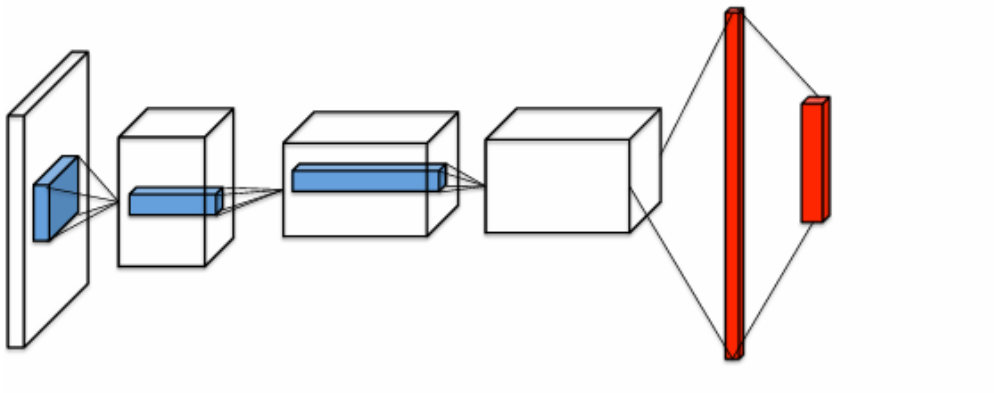


Figure 2.3 Deep Q-Network (DQN) architecture: input state (e.g., traffic features) is processed through convolutional and fully connected layers to output Q-values for each action. [29].

DQN has demonstrated a significant capability in minimizing vehicle delays and increasing throughput compared to fixed-time or rule-based controllers [4][7]. Even though the straightforward DQN algorithm has demonstrated worthwhile promise, DQN is inherently flawed in two major respects:

1. Overestimation bias — Due to a maximization step in Q-value updates, DQN tends to overestimate action values. As such, all policies derived from DQN may also be biased, leading to instability.

2. Sample efficiency — DQN generally requires considerable amounts of samples or iterations to reach convergence, which can be computationally expensive in a simulation setting.

To address these deficiencies, a few improved versions of DQNs have been proposed. Perhaps the most relevant for traffic signal control is the Dueling Deep Q-Network which separates V(s) from A(s,a) for Q-value estimate computation. This architecture enables the network to use the benefit of a state evaluation without presuming the relative differences of actions are meaningful. Meanwhile, Dueling DQN can improve stability and convergence in noisy or sparsely rewarded environments [29]. Figure 2.4 illustrates this structure.

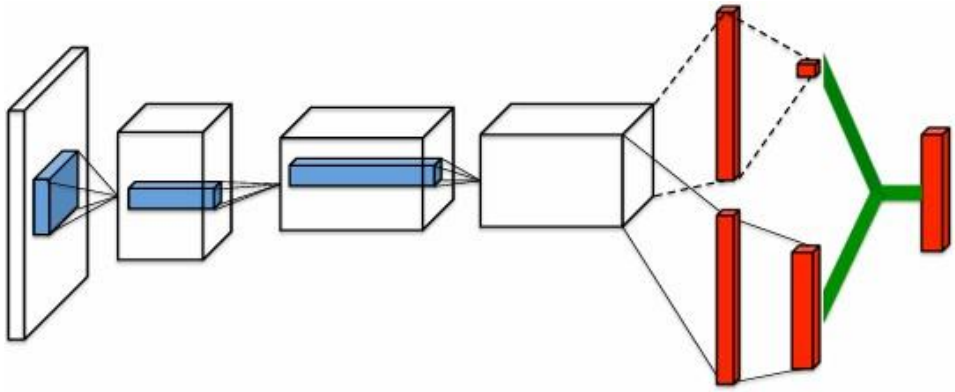


Figure 2.4 Dueling DQN architecture: shared feature extractor splits into Value stream V(s) and Advantage stream A(s,a), which are combined to compute action-value estimates Q(s,a). [29].

Other add-on techniques include Double DQN, which mitigates Q-value overestimation by separating evaluation and action selection, and Prioritized Experience Replay, which boosts the efficiency of sampling by replaying transitions with greater expected learning potential [8][9]. These developments have become commonplace in ideas related to DRL for traffic signals controller work and form the basis of good controllers for more robust and scalable situations.

To isolate the individual contributions of each architectural improvement, this project conducts a controlled ablation study comparing four DQN variants under identical conditions — the same composite reward function, the same hyperparameters, the same

SUMO scenario, and the same number of training episodes. The four models are: (1) VanillaDQN, which uses a single Q-head and the standard DQN target (target network selects and evaluates, causing overestimation bias); (2) DoubleDQN, which retains the single Q-head but applies the double Q-target update rule to reduce overestimation; (3) DuelingDQN, which introduces the value and advantage stream decomposition but uses the vanilla target; and (4) DuelingDoubleDQN, which combines dueling streams with the double target update. This 2×2 design isolates the contribution of the double update rule and the dueling architecture independently, enabling a direct attribution of observed performance differences to each component.

2.1.3 Advanced DRL Techniques After 2022

Although DQN and variants like Double DQN and Dueling DQN provided a springboard for powerful signal optimisation strategies, contemporary research has built upon these developments with architectural improvements that have enhanced performance further, especially in the context of more complicated and large-scale urban networks. These modifications aspire to tackle the challenges of scalability, state representation, and long-horizon decision making, as DRL-based signal controllers make the transition from isolated intersections to city wide implementation.

This was accomplished through one major contribution, the application of convolutional and attention-based neural structure in traffic signal control. These functional models can capture space dependence of lanes and approaches, as well as time dependence of traffic flow through conditioning of the convolutional layers or attention on the representation of traffic states. An example of this would be using attention-enriched DQNs to discretely weight meaningful features (queue lengths on crucial approaches for example) and ignore noise, thus leading to improved performance and robustness to shifts in read traffic conditions [8].

The other line of work has been the review of graph neural networks (GNNs) and reinforcement learning (RL) for multi-junction control. Specifically, in these methods, a junction is thought of as a node in a graph, and the fitness of traffic movements in between adjacent junctions is defined by an edge [10]. GNN-based DRL agents have

the capability of learning both local policies and coordinated behaviours through communication of messages within the GNN structure. This technology improves scalability and coordination in a corridor or network-wide traffic optimization sense, which is a long-standing limitation of multi-junction decentralized RL controllers.

Another avenue of development is with sequence models such as the decision transformer, a reclassification of reinforcement learning as a sequence prediction task that uses the sequence transformer [12]. These same models can implicitly learn further temporal dependencies through conditioning and create policies that introduce a trade-off between short term throughput and long-term efficiency. While the application of this framework in the domain of traffic modelling is still relatively new, the preliminary results suggest that based on initial testing decision-transformer based controllers were preferable over traditional DQN-like architecture in dealing with highly stochastic or partly observable environments.

Figure 2.5 illustrates a convolutional Dueling DQN architecture applied to traffic signal control [8]. Here, a shared convolutional feature extractor processes raw or structured traffic state data, after which the network branches into two streams — one estimating the state-value function and the other estimating the advantage function. The results are then recombined to produce Q-value estimates. This architectural design highlights the trend of combining value decomposition (Dueling) with more sophisticated feature extraction (CNNs, attention) to improve both convergence stability and sample efficiency in complex urban scenarios.

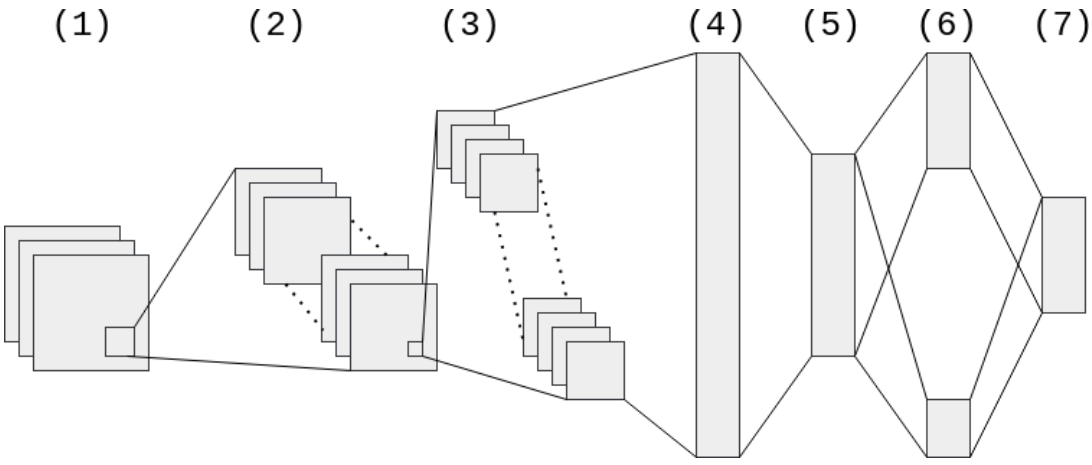


Figure 2.5 Convolutional Dueling Deep Q-Network (DQN) architecture for traffic signal control. [4]

Despite their potential positives, it must also be noted that various architectures add substantial complexity and computational requirements to the model. Additionally, many of these existing approaches are focused on multi-agent, multi-intersection networks, while this project is focused on a single-intersection scenario. Thus, while attention based DQN, GNN-augmented controllers, and Transformer based RL are important ideas in the evolution of intelligent traffic management, the focus will be on a controlled comparison of Q-learning, standard DQN and Dueling DQN, so that this study can more clearly isolate the structural benefits of Dueling DQN in a realistic but comparatively simple experimental context.

2.1.4 Benchmarking and Real-world Deployments

The assessment of traffic signal controllers based on DRL relies heavily on benchmarking as part of experiments in realistic but controllable environments. Benchmarks like field trials are costly and disruptive, if not useful. Replicating representations in field trials within this and across studies is problematic, so most studies adopt microscopic traffic simulators as their experimental testbed. SUMO [16] and CityFlow [14] are arguably the most broadly adopted and supported simulators, as both options are well-developed to facilitate exploration of desired intersection geometries, vehicle dynamics, and traffic demand patterns, and include APIs to allow real-time interaction between the traffic environment and the trained learning agent. As a traffic signal controller and modelling tool, SUMO has drawn popularity as an open-source, highly flexible, support multimodal traffic management including different traffic modes (bicycles, pedestrians, public transport), representation of left- hand traffic (where required) and for four-way negotiations making it a useful option for demonstration in regions such as Singapore, Malaysia, or the UK. In the figure 2.6, it exemplifies the types of systems represented in the SUMO context with a graphic from a SUMO simulation illustrating four-way impulse LOS protocol with left-hand traffic behaviours [11]. Contrastingly, CityFlow provides extremely rapid simulation across large-scale networks and is commonly used in studies targeting multi- intersection, city-

level network experimental backgrounds [14].

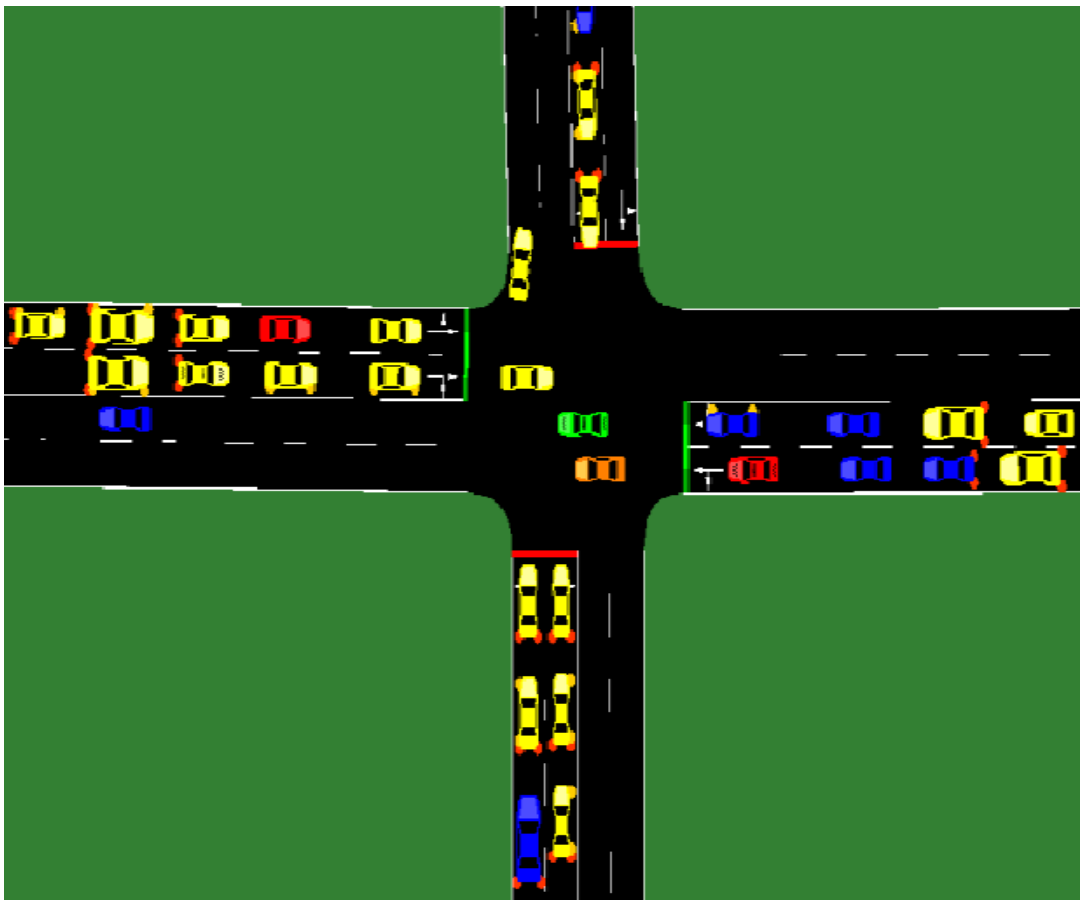


Figure 2.6 A SUMO simulation of a four-way intersection with left-hand traffic dynamics. [7]

Across all these platforms, DRL-based controllers have demonstrated significant improvement in comparison to fixed-time and actuated baselines. Reported improvements include reduced average vehicle delay, total queue length, number of stops, and fuel consumption, and an increase in throughput and total intersection capacity [7][14][16]. The simulation studies illustrate that there is potential for DRL to provide significant improvements traffic efficiency; these studies should, however, also highlight that standardized benchmarks need to be used to provide a comparable measure to other studies.

Although there has been exciting progress in simulation there are still very few instances when DRL controllers have been deployed in the real-world. To date, two

noteworthy examples are the Hangzhou pilot (2022) and in Singapore. The Hangzhou pilot was a real-world application of an offline DRL controller deployed at live intersections and used CityFlow to develop the controller through simulated training, with preliminary results indicating the DRL controller produced a 17% increase in peak-hour throughput relative to existing adaptive control systems [15]. The Singapore example was more extensive as it used DRL agents with roadside sensors that updated the signal timings every five minutes, allowing for a less restrictive and more efficient signalized intersection system [11]. The Hangzhou and Singapore pilots provide solid validation that DRL is capable of operating in a real-world context, however, issues remain around coverage, data latency, model interpretability, and legacy control.

Scientists are also creating IoT-based frameworks leveraging real sensor data streams with cloud-hosted DRL agents that introduce new possibilities for flexible and scalable signal control [15]. However, there are issues related to system latency, cybersecurity, and interoperability with existing traffic management methods with this type of approach. Consequently, we find that most of the DRL literature is still using simulations such as SUMO in a safe, flexible, and reproducible manner to develop algorithms and make controlled comparisons.

In conclusion, benchmarking through simulation has shown that DRL could be an effective method for traffic signal control and initial pilot deployments in Hangzhou and Singapore have shown real benefits in practice. However, the disparity between simulation and operational deployed systems is still a large research frontier, specifically when it comes to robustness, safety assurance, and the implementation of real systems at scale.

2.2 Limitation of Previous Studies

Research in deep reinforcement learning (DRL) for traffic signal control has provided exciting simulation results, but Springer consistently highlights multiple limitations which call into question the validity of single intersection SUMO-based trials. These include explicit generalization failures, intrinsic reward design issues, algorithmic instability with benefit to drivers being an unknown, sim-to-real transfer error, and

reproducibility failure. Each limitation is listed and explored below with references and were pertinent, illustrative summaries.

2.2.1 Lack of Generalization Across Environments

However, it is important to address the main drawback of DRL for traffic signal control, which is its poor transfer across environments [17]. Training of most DRL models is conducted under specific traffic conditions or a specific simulation, and there is often performance degradation when applied to a new location [9]. DRL agents often demonstrate poor adaptation ability when transitioned to a different intersection or city. Therefore, the real-world uptake of DRL and other deep learning-based approaches for traffic signal control is limited.

An example is a study of a DQN model trained for a grid network, which had limited performance on a random shape downtown intersection with longer queue lengths and unstable signal phases [9]. This study also highlights the inability of DRL models to capture different traffic flow patterns, lane hierarchies, or road topology.

Generalization is further impacted because traffic is stochastic and drivers behave with diverse behaviours [1], [17]. Real world scenarios are diverse and variable due to factors such as the weather, accidents, catastrophic events, and social events - for many scenarios, we do not have a limited scenario chosen as we would in a simplified simulation approach. As a result, DRL models trained solely on simulation tend to apply poorly to generalization due to real world variations [18].

A different experiment concluded that DRL agents trained with fixed traffic densities, when tested on varying volumes or turning ratios, failed unless the use of either domain randomization or transfer learning [18]. It has been proposed that curriculum transfer learning - in which the models are trained on incrementally higher complexity scenarios - can resolve adaptability in heterogeneous environments. However, while these strategies help with adaptability, they also increase the complexity of the system and do not yet providently generalised. In table 2.1, a collection of representative studies is

summarised where DRL based controllers could not generalise over environments with an observed consistent gap between performance in trained conditions and deployment conditions.

Training Environment	**Deployment Environment**	**Outcome**	**Citation**
Grid network (simulation)	Irregular downtown intersection	Increased queues, unstable signals	[9]
Fixed traffic density	Varying volumes / turning ratios	Severe performance drop without transfer/domain randomization	[18]
Simplified simulation (no weather/events)	Real-world traffic with stochastic factors	Poor adaptability, degraded performance	[1], [17]

Table 2.1 Comparison of Training Environment

In summary, solid generalization remains a significant obstacle for DRL-based traffic signal control. The goal of overcoming this limitation is important if urban deployments are going to be able to operate sufficiently over time without frequent retraining or over-tuning the system.

2.2.2 Reward Function Design Constraints

The design of reward functions is an essential aspect of DRL-based traffic signal control [1]. Reward functions define goals for the learning system, and they also significantly influence the agent's behaviour. Most research uses relatively simple reward structures, where the focus is largely on vehicle delay and spatial queue length [7]. Those variables are relevant; however, solely focusing on them ignores other aspects such as pedestrian security, environmental concerns, and equitable distribution of traffic volume [3]. Such a narrow view can cause unwanted results, i.e., optimising priority for the major street may result in long delays and favour uncertain or dangerous conditions for pedestrians and vehicles on the minor street [17].

Additionally, reward functions can be too static. Traffic movement patterns are usually

dynamic, and many unforeseen aspects such as accidents, road closure (i.e., construction projects), public events, or changes in driver behaviour can have an influence. If the reward function does not consider those shifts, or act appropriately, it may fail to direct the agent to make reasonable, timely, or effective decisions. For instance, a basic reward function is unlikely to account for emergency vehicle priority, nor is it likely to integrate sudden increases / high demand for priority for pedestrians, thus limiting the efficacy of the system [17].

One example of a weighted reward function can be expressed as:

$$R = -(w^1 \times D + w^2 \times Q + w^3 \times P + w^4 \times E)$$

New research has proposed multi-objective and adaptive reward functions to evaluate algorithms in place of the traditional single-objective reward functions. For example, reward functions including a mixture of vehicular flow and pedestrian flow have shown success in controlling traffic efficiently [15]. Adaptive rewards that dynamically weight each objective depending on the current state of traffic have also shown promise. However, the increased complexity of the multi-objective and adaptive rewards leads to larger amounts of data needed and greater computational costs that can limit the deployment of large city scale DRL-based traffic signal control systems. Table 2.2 compares the types of reward functions used in a DRL-based traffic signal control system.

Reward Function Type	**Features Considered**
Simple Reward	Vehicle delay, queue length
Multi-Objective Reward	Delay, queue, emissions, pedestrian flow
Adaptive Reward	Dynamically adjusted weights based on traffic/pedestrian patterns
Safety-Aware Reward	Pedestrian wait time, conflict zones, accident likelihood

Table 2.2 Comparison of reward function designs in DRL-based traffic signal control systems.

2.2.3 Instability, Overestimation, and Sample Inefficiency in DQN

Although deep Q-networks (DQNs) have shown potential for adaptive traffic signal control, a limiting factor is the instability of the training process and the bias in overestimation. Ordinary Q-learning updates always employ the maximisation operator :

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma \boldsymbol{max} Q(s',a') - Q(s,a)]$$

when Q-values are noisy [29]. This is done in traffic environments where the goal guided policies can demonstrate unstable learning dynamic where Q-values diverge or oscillate rather than develop stable policy [17]. As a result, controllers can demonstrate erratic handling whereby they will rapidly switch phases or can heavily bias priority towards certain approaches resulting in increased congestion.

Another challenge is sample inefficiency. DQNs requires above many interactions to learn a useful policy which increases the expense and the time of training in realistic simulations. Experience replay may somewhat offset the inefficiencies, but naive sampling would scale down the efficiency of computations as unnecessary interactions would be wasted on transitions that are mostly redundant [8]. The effect of hyper-parameter tuning sensitivities could be further confounded by studies that do not present seed-averaged results which really decreases reproducibility [17]. In essence, performance differences across studies, is due to a tuning bias with little to no real improvements in the algorithm.

To alleviate these challenges, multiple solutions were proposed by researchers; Double DQN, which decouples action selection from evaluation to eliminate over estimation bias, and Dueling DQN, which decouples state-value estimation from advantages for added stability [29]. Also, prioritised experience replay may help sample efficiency via experience that has higher temporal-difference (TD) errors is replayed [8]. With target

networks, which are updated slowly, this theoretically prevents divergence from estimating the action-value function [29]. Even with neither of these advances, researcher in traffic-signal control studies have not been consistent or missing in studies on existing DQN methods to facilitate comparative benchmarking; and even this is disheartening to longitudinal comparisons, and direct comparisons difficult to make.

Method	Mechanism	Benefit	Limitation	Citation
Double DQN	Decouples selection/evaluation	Reduces overestimation bias	Slightly slower convergence	[29]
Dueling DQN	Value/advantage decomposition	More stable learning	Higher network complexity	[29]
Prioritised Replay	Replay high-TD-error transitions	Improves sample efficiency	Introduces sampling bias	[8]
Target Network	Slowly updated target Q-network	Reduces divergence	Requires careful update tuning	[29]

Table 2.3: Stabilisation techniques for DQN-based traffic signal control

This paper includes Double DQN and Dueling DQN as stabilisation techniques and averages proteins results in case of many random seeds to enhance reproducibility and fairness between them.

2.2.4 Simulation-to-Reality Transfer Gap

Experiments conducted with DRL within virtual environments has achieved very high success rates, notably, simulation is a viable and effective means to devise traffic signal control plans as was previously mentioned. Despite this, it is problematic to take DRL models which were developed under a controlled simulation environment and simply deploy them in the unpredictable complexity of the real world; that phenomenon is commonly known as the simulation-to-reality gap [17].

The sim-to-real gap is due to the mismatch between the simplified simulation setting and the real-world dynamic traffic interactions. There are many factors that will affect the re-world traffic interactions including noise from traffic sensors, not uniform driver behaviours, and external conditions such as inclement weather. It is often the case, the only assumption made when training a DRL model will not be accurate, the model operates in simulation much better than its re-deployment; thus, the model will have poor performance or instability or even unsafe behaviours.

In order to mitigate this limitation, several strategies have been proposed. For example, how to adapt actions identified in the simulation setting to ensure that they better suit real-world traffic dynamics and traffic regulations grounded action transformation (GAT) [18]. More sophisticated frameworks have recently been developed like an uncertainty-aware GAT (UGAT), accounting for stochastic variability and safety constraints, and they have demonstrated a more reliable transfer of knowledge across domains.

Figure 2.7 illustrates the comparison between simulated and real-world DRL performance (left), and the architectural differences between vanilla GAT and Uncertainty-aware GAT frameworks (right).

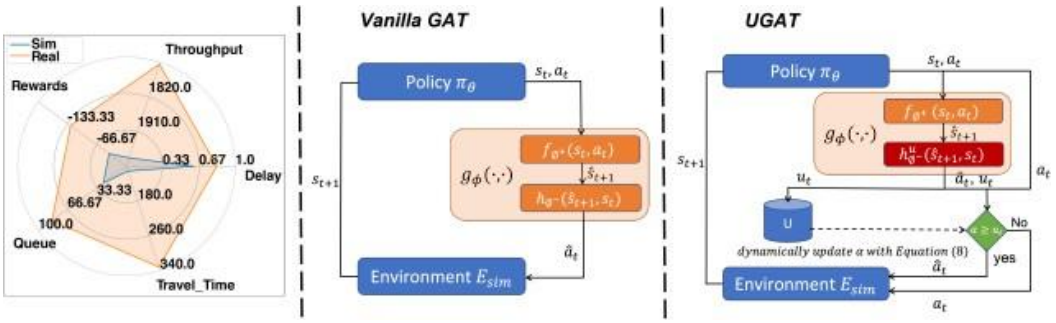


Figure 2.7 Comparison between simulated and real-world DRL performance (left), and architectural differences between vanilla Grounded Action Transformation (GAT) and Uncertainty-aware GAT (UGAT) frameworks for bridging the sim-to-real gap in traffic signal control. [25]

Recent research has also examined the potential of large language models (LLMs) as a means of positively addressing the sim-to-real gap. Specifically, in their research on

traffic signal control, PromptGAT uses LLMs to contextualize many of the complex factors that exist in the traffic signal control context such as, variations in weather, spikes in demand, which would enable the model to adapt better [17]. These procedural models remain compelling; however, LLM-models would increase the computational complexity of the problem along with needing large amounts of real-world data for subsequent fine-tuning.

In conclusion, the sim-to-real gap remains a significant barrier to the real-world applicability of DRL-based traffic signal control. While there have been some attempts to more positively address this gap, such as UGAT, and LLM-inspired approaches at least help to make some progress, reliable, safe and generalisable performance in real-world, dynamic traffic situations remain open research that still needs to be addressed.

2.3 Proposed Solutions

All the above limitations have prompted the invention of state-of-the-art solutions. In this section, the theoretical background and methodological strengths, behind the current research, are explained.

2.3.1 Rationale for Using Dueling Deep Q-Network (Dueling DQN)

Deep reinforcement learning (DRL) has shown promise for traffic signal control, with Deep Q-Networks (DQNs) perhaps the most popular DRL algorithm. However, standard DQNs may have issues with over-optimism in action values, instability during learning, and convergence rate. For the application of traffic signal control this can lead to sub-optimal decisions in an ever-changing environment.

Dueling Deep Q-Networks (Dueling DQNs) were introduced to address these concerns. The main concept was to have the advantage function and value function in DQNs separated into two streams. This helped associate their respective contributions, allowing the model to learn which state has value without needing to estimate the

nominators of the A(s,a) every possible action for every possible state. The net result is that Dueling DQNs are more reliable and learn faster than standard DQNs, while also improving generalisability with previous states.

Figure 2.8 presents the Dueling DQN architecture. The Dueling DQN splits into two streams after the dense layers shared by both streams, with one stream computing the state-value function and other stream estimating the advantage function. Q-values are computed using these two outputs, helping the network to learn as it relates to the importance of a state, but also factor in variation that depends on actions. This unique structure is one way to promote more stable learning and is ideal for this application in adaptive traffic signal control [14].

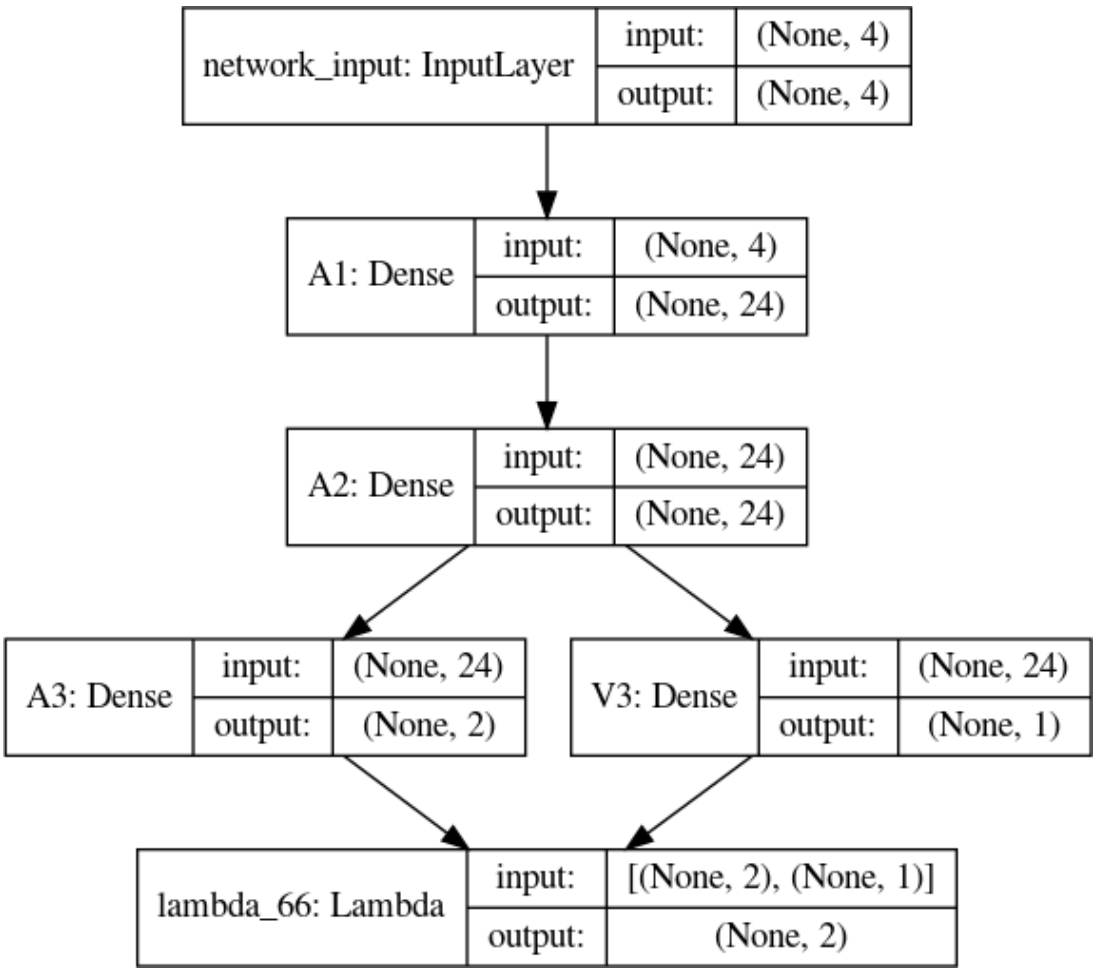


Figure 2.8 Architecture of a Dueling Deep Q-Network (Dueling DQN). [26]

2.3.2 Algorithmic Stabilisation

Section 2.2 identified the instability and overestimation bias learning of standard DQN as a core challenge, one that could lead to bad convergence and hence unreliable traffic control decisions. To help with this, some states in the literature have been explored algorithmic stabilisation techniques such as Double DQN, Prioritised Experience Replay (PER), and various combinations, for e.g. Dueling DDQN.

The Double DQN framework was established mainly to reduce the problem of Q-value overestimation of standard vanilla DQN, especially by disentangling action execution and action value estimation from the execution. In the context of traffic signal control, this approach is employed because of some improvement in stability and robustness of the learned policies. Xu, Wang and Li [26], indicated in their work that a cooperative traffic control model using Double DQN had more stable behaviours in uncertain and stochastic demand environments, and more reliable to perform better than DQN, traditional methods prior to the research.

Prioritized Experience Replay (PER) is another comparative academic technique to stabilize and enhance learning with DQN. PER reassigns big temporal-difference error experiences with greater sampling probabilities, which allows the agent to ensure it learns more firmly with the environmental transitions most important for reducing estimation errors. [23] applied PER in a deep reinforcement learning urban traffic signal controller and reported faster convergences and lower average queue lengths using PER rather than uniform replay.

Another avenue of research may also be to combine these approaches, which can yield greater robustness. For example, the Dueling and Double DQN approach was shown to speed up and reduce policy variance in complex traffic scenarios. Dueling DDQN agents distinguish value estimates and advantage estimates and better separate the importance of states based on advantage estimates. And although priority experience replay (PER) allows Dueling DDQN to perform more efficaciously, it brings additional stability because of the stochastic nature of the traffic flow employed by an agent to learn from [26],[23].

Ultimately, these algorithmic stabilisation methods, are targeted at the problems of instability, overestimation and sample inefficiency as given in Section 2.2, and represent meaningful contributions to the continued development of reliable and scalable traffic signal control agents.

Technique	Reported Benefit	Source
Double DQN	**Reduces Q-value overestimation, improves robustness under dynamic traffic**	**[26]**
Prioritized Experience Replay (PER)	**Enhances sample efficiency, accelerates convergence, reduces queue lengths**	**[23]**
Dueling DDQN (with PER)	**Combines state-value and action advantage estimation with efficient replay, improves stability across noisy flows**	**[26],[23]**

Table 2.4 Stabilisation methods for traffic signal control.

2.3.3 Realistic Training via SUMO and Data Integration

Training DRL models for traffic signal management in real-world situations is fundamentally unfeasible due to safety issues, costs, and the variability of characteristics in urban traffic patterns. Therefore, microscopic simulators, such as the SUMO (Simulation of Urban Mobility) simulator, are valuable environments for designing and testing DRL-based traffic management approaches.

Figure 2.9 shows the SUMO graphical user interface (GUI) simulating a signalised intersection with four approaches. These simulations give the researcher the ability to control road networks, signal phases, and vehicle behaviours that allows them to test and remediate a DRL algorithm before being implemented in the real world [16].

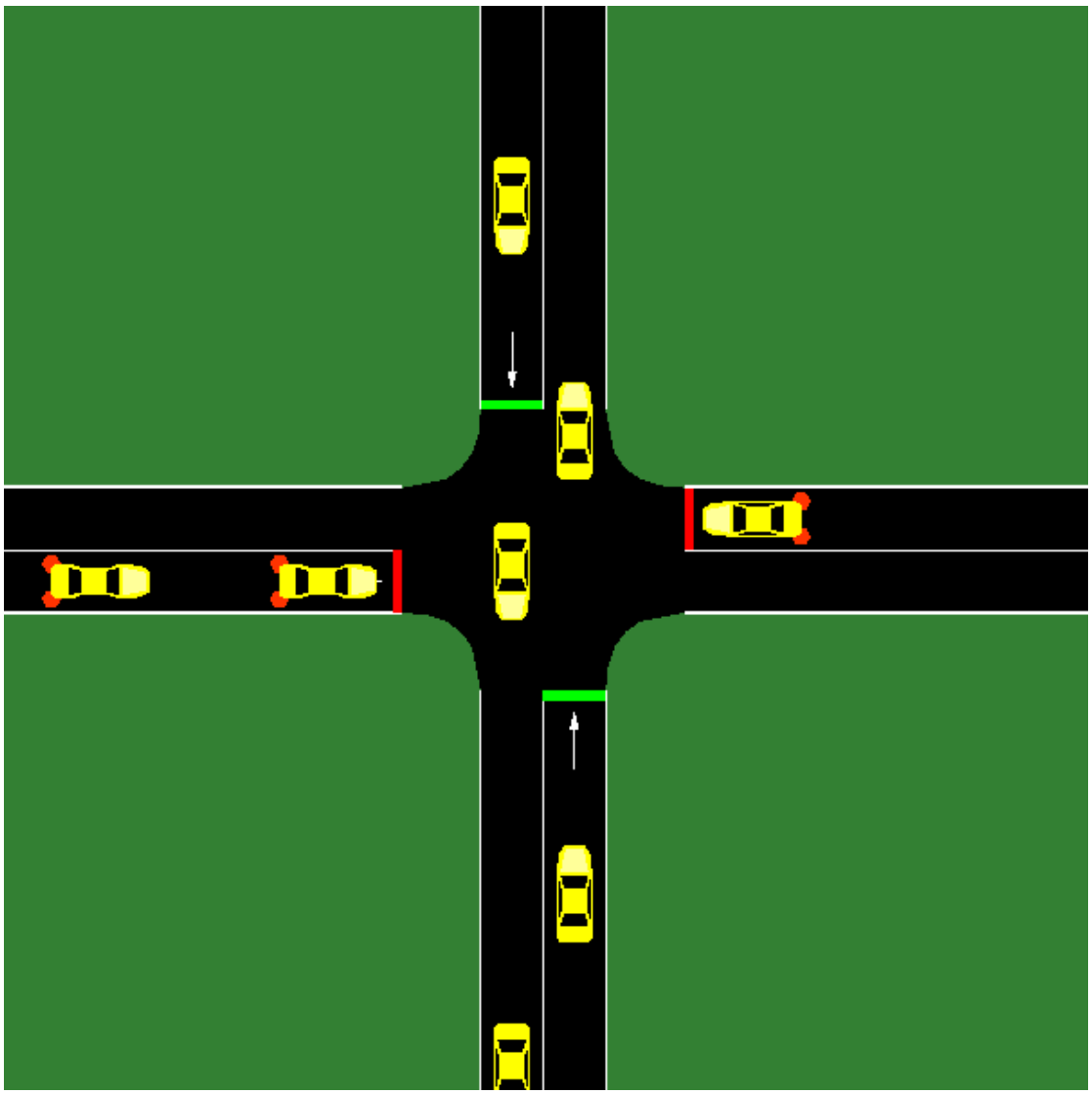


Figure 2.9 SUMO GUI displaying a signalised intersection with four lanes. Adapted from SUMO documentation [16].

Through its flexible structure, it can describe many details of traffic scenarios, such as inhomogeneous vehicle types, multi-phase signal plans, and extensive urban road networks. Based on SUMO, the SUMO-RL framework adds enforcement learning interfaces and provides research with an adaptable setting for developing traffic signal control agents.

Table 2.5 The major disparities between SUMO and SUMO-RL are in the way that they are suited to DRL-based experimentation.

Feature	**SUMO**	**SUMO-RL**

Simulation Engine	Microscopic traffic simulation	Built on top of SUMO
DRL Support	No native DRL	Integrates Gymnasium, PettingZoo for RL training
Agent Support	Manual or external control	Supports single-agent and multi-agent learning
Reward Customisation	Not built-in	Fully customisable via Python
Realism & Flexibility	Very high	High (inherits SUMO's realism with DRL enhancements)

Table 2.5 Comparison of SUMO and SUMO-RL features for DRL-based traffic signal control

The SUMO-RL framework supports both single-agent and multi-agent modes and is also compatible with modern RL libraries. For instance, Gymnasium and PettingZoo, giving researchers the ability to include custom state representations and reward functions suited to their specific traffic conditions. Recent studies using the SUMO environment with advanced DRL architectures, (Xu, Wang, and Li [26]), demonstrates the ability to significantly improve the stability and effectiveness of traffic signal controllers in multi-agent settings. These integrations indicate the usefulness of SUMO as a safe, yet realistic platform for developing future AI-based traffic control research.

2.4 Summary

This chapter reviewed the background, limitations, and proposed solutions regarding reinforcement learning methods for adaptive traffic signal control. In Section 2.1, we introduced the move from traditional fixed-time and actuated signal control systems to deep reinforcement learning models such as DQN and Dueling DQN. In Section 2.2, we addressed the main limitations of these approaches, including instability and overestimation bias, sensitivity to reward design, weak generalisation, difficulties in sim-to-real transfer, and reproducibility issues.

To examine these challenges, Section 2.3 discussed promising methodological improvements raised in the literature. The Dueling DQN architecture was noticed and presented as promising potential to address instability and overestimation issues introduced by the duality of state and action evaluation through separating the state-value and advantage estimate. Algorithmic stabilisation techniques implemented in Dueling DQN, such as Double DQN and Prioritized Experience Replay, were noted to also help the learning agent to further improve training efficiency and robustness. Realistic simulation environments, demonstrated in studies such as SUMO and SUMO-RL, present a safe, but realistic, and flexible domain for experimentation. In summary, it is encouraging that developments in current research go beyond Baseline DQN, as noted in previous sections, and address several of the challenges outlined above.

To draw a conclusion to Section 2 the literature suggests Dueling DQN built on a robust training protocol and realistic simulation environment, will provide a meaningful starting point for adaptive traffic signal control. These findings feed directly into the design of the methodology for the current study, which is to implement Dueling DQN with a composite reward structure, in a SUMO environment. The next chapter provides a description of the methodological framework, experimental setup, and evaluation procedures used, which captured the proposed approach.

CHAPTER 3 - Proposed Method/Approach

3.1 Methodology

This experimental project is a readaptation of the traffic signal controller using deep reinforcement learning called Dueling Deep Q-Network (Dueling DQN), which is designed and gauged in the SUMO traffic simulation environment. Python is the language of choice in this implementation, and the developer has utilized PyTorch as a deep learning backend and TraCI (Traffic Control Interface) as the interface for the control and real-time monitoring of SUMO environment. The methodology describes the entire process as explicit steps, and all the experiments are relying on the files TrafficLight.sumocfg, TrafficLight.net.xml, TrafficLight.add.xml, and TrafficLight.rou.xml. Section 3.3 contains the exact mathematical representation and diagrams.

PyTorch: Among the most widely used open-source frameworks for deep learning is PyTorch. It is mostly used in the establishment, and training of neural networks. It enables a user to construct the Dueling DQN architecture with the description of network layers, the implementation of the activation functions and the parameter updating based on the learning using the gradients. PyTorch is ideally suited to the needs of reinforcement learning with the dynamic nature of the computation graph since a training process is performed on each state transition and update.

TraCI: The Traffic Control Interface (TraCI) is SUMO's Python API that enables external programs to interact with the simulator during runtime. With TraCI, the adaptive controller has access to traffic-related information (queue length, waiting time, vehicle positions) and can use it to send commands to dynamically change the traffic-light phases. Such a bi-directional communication enables the reinforcement learning agent to feel the state of the traffic, act (e.g., by switching the signals), and obtain feedback (through the updated traffic state), which the agent can use to learn and update its strategy.

The methodology consists of the following steps:

Step 1: SUMO scenario and environment setup

Overview — The SUMO scenario is defined by TrafficLight.sumocfg, which loads the network

TrafficLight.net.xml, the additional detectors file TrafficLight.add.xml, and the route flows TrafficLight.rou.xml. The reinforcement-learning controller is focused on the traffic light of Node2 or the centre junction of the network. The training script must start SUMO with the TrafficLight.sumocfg file that the report documents so TraCI connects to the intended scenario and detector IDs.

Traffic-light logic (tlLogic) — The traffic light for Node2 in the net.xml defines **five phases**: a 42s E/W green phase, a 3s yellow transition, a 37s N/S green phase (which includes a pedestrian signal), a 5s N/S-only phase, and a 3s yellow transition. This is the **default fixed-time controller** defined in net.xml. The RL agent overrides this at runtime using traci.trafficlight.setRedYellowGreenState() to apply one of three discrete phases defined in the code.

Route flows and direction meaning — Route flows in TrafficLight.rou.xml give vehicle demand by origin→destination. The approach totals are: EC (East→Centre) total = 499 veh/hr (125→CN, 240→CW, 134→CS), NC (North→Centre) total = 460 veh/hr (85→CW, 105→CE, 270→CS), WC (West→Centre) total = 450 veh/hr (200→CS, 100→CE, 150→CN), SC (South→Centre) total = 400 veh/hr (175→CN, 100→CW, 125→CE). The total network demand is 1,809 veh/hr. The training scenario (medium) uses adjusted flows of approximately 1,809 veh/hr defined in TrafficLight_rou.xml.. Two-letter IDs encode origin→destination: EC means East → Centre(vehicles arriving from the east), CS means Centre → South (leaving Centre to the south), etc. If you assume evenly split lane flows (each incoming edge has two lanes), divide each approach total by 2 to estimate per-lane veh/h (e.g., EC ≈ 857.5 veh/h per lane).

Detectors and parameters — Lane-area detectors are declared in TrafficLight.add.xml with IDs EC_0, EC_1, NC_0, NC_1, SC_0, SC_1, WC_0, and WC_1. These are lane-area detectors that are mounted over the incoming lanes and these detectors are employed to determine queues and waiting times by approach. Every detector specified in the add file has pos, length and a period attribute; Detector positions vary by approach due to different edge lengths: EC at pos=115m, NC at pos=90m, SC at pos=40m, WC at pos=80m. Note thatperiod="300.00" causes detector CSV outputs to be produced only every 300 s — for per-step control you should either query detectors each control step via TraCI (recommended) or change the period to a

smaller value (e.g., 1.0 s) if you want frequent CSV logs.

Simulation settings — The sumocfg file references TrafficLight.net.xml, TrafficLight.rou.xml, and TrafficLight.add.xml. Simulation duration and timing are controlled by the Python training script via CONFIG settings: max_sim_time = 3,600 seconds per episode, step_length = 0.1s, and yellow_duration = 3s. In the methodology, the controller works with a control step_length of 0.1 s; ensure SUMO is started with the same step length (or your controller advances the simulation using traci.simulationStep() with the same timestep) to ensure timing of the green/yellow periods and detector values are consistent.

Step 2: Data capture and preprocessing

At each step the simulation provides detector data:

The controller gathers measurements of the traffic on the eight lane-area detectors specified in TrafficLight.add.xml (EC_0, EC_1, NC_0, NC_1, SC_0, SC_1, WC_0, and WC_1). at each simulation step. These detectors span the lanes approaching on each end. Queue counts are retrieved via traci.lanearea.getLastStepHaltingNumber(detector-id) which gives the number of vehicles in detector-area that have less than 0.1 m/s velocity. Waiting times are determined by retrieving the vehicle IDs per detector using traci.lanearea.getLastStepVehicleIDs() and adding together the waiting times using traci.vehicle.getWaitingTime(vehID). Detector values from lanes of the same approach are aggregated to give per-direction totals, for example EC_0 + EC_1 → qE, wE for the east approach. The state vector passed to the agent is a 13-dimensional vector defined in Equation 1, comprising a one-hot phase encoding, four directional vehicle queue counts, four directional vehicle waiting times, a pedestrian queue count, and a pedestrian waiting time. In case a detector provides missing data, the implementation replaces the missing data with zeros and logs a warning to ensure stability throughout training. Continuous per-step records of queue counts, waiting times, and vehicles passed are kept and subsequently averaged to calculate episode statistics including mean waiting time, mean queue length and throughput by direction.

Step 3: Agent design and action definition

The reinforcement learning agent is implemented as a Dueling Double Deep Q-Network (Dueling

DDQN). The neural network consists of two fully connected hidden layers with 256 neurons each. Layer normalisation (LayerNorm) is applied after each linear transformation and before the ReLU activation, stabilising the internal activations across the varying input magnitudes that arise under different traffic demand levels. Shared layer weights are initialised using Xavier uniform initialisation; the value and advantage output heads are initialised with near-zero uniform weights in the range [−0.01, 0.01] to prevent the large Q-value spike that typically destabilises early training.

The Dueling architecture splits into two parallel streams after the shared layers. The value stream $V(s; \theta)$ outputs a scalar estimating the overall quality of the current traffic state independent of any specific action. The advantage stream $A(s, a; \theta)$ outputs one value per action, estimating the relative benefit of choosing that action over the average.

Step 4: Training loop

The agent is trained using experience replay with a buffer capacity of 50,000 transitions. Learning begins once the buffer contains at least 1,000 transitions, ensuring sufficient diversity in early minibatches. At each learning step, a minibatch of 128 transitions is sampled uniformly from the buffer, breaking the temporal correlation between consecutive training updates that would otherwise bias the gradient estimates.

The Adam optimiser is used with an initial learning rate of 1×10^{-4}. A Cosine Annealing learning rate scheduler (CosineAnnealingLR) smoothly decays the learning rate from 1×10^{-4} to 1×10^{-6} over the full 1000-episode training run. This progressive reduction prevents disruptive weight updates in late training when the policy is near convergence, reducing oscillation around the final learned behaviour.

Exploration follows an ε-greedy strategy. ε begins at 1.0 (fully random) and decays by a factor of 0.9970 per episode, reaching the minimum value of 0.05 at approximately episode 1000. A minimum of 5% exploration is retained throughout training to preserve responsiveness to unseen traffic patterns. Gradient clipping with a maximum norm of 1.0 is applied after each backward pass to prevent weight divergence.

The target network is updated by soft copy with $\tau = 0.005$ at every learning step, continuously blending 0.5% of the online policy network's current weights into the target network. This produces smoother Q-value targets than the periodic hard copy used in earlier DQN variants, reducing sudden destabilising shifts in the learning objective. Model checkpoints are saved every 50 episodes; the checkpoint at episode 1000 is used as the final evaluation model.

Step 5: Evaluation and reproducibility

Some of the performance measures used to assess the trained agent are the mean waiting time per vehicle, and mean queue length, throughput by direction, cumulative episode reward, and training loss. These metrics present a moderated perspective of traffic efficiency and stability of learning. It is compared against three ablation baseline models — VanillaDQN, DoubleDQN, and DuelingDQN — across low, medium, and high demand scenarios to evaluate the contribution of each architectural component, which manipulates signals depending on measurements of traffic demand. Under varying traffic conditions (low, medium, and high demand conditions) to assess the robustness of the reinforcement learning approach, comparisons are made.

To make the statistical reliability, All scenarios use a fixed SUMO random seed, producing deterministic results across test episodes. Standard deviation is therefore zero across all 5 test episodes per scenario, and results are reported as exact mean values, giving the central performance trends as well as variability between runs. To enable reproducibility, all SUMO configuration files, route definition, detector specifications, random seeds, model checkpoints, and training

logs are saved. Using a fixed --seed in SUMO and specifying random seeds in Python, NumPy, and PyTorch is also suggested to allow experiments to be reproducible with exact accuracies.

Key hyperparameters of the final implementation:

Hyperparameter	Value	Notes
Replay buffer capacity	50,000 transitions	Circular FIFO buffer
Minibatch size	128	Sampled uniformly per learning step
Discount factor γ	0.9771	Tuned; balances short and long-term reward
Initial learning rate	1×10^{-4}	Adam optimiser
LR scheduler	CosineAnnealingLR	Decays to 1×10^{-6} over 1000 episodes
Epsilon (start)	1.0	Pure random exploration at episode 1

Hyperparameter	Value	Notes
Epsilon (minimum)	0.05	5% exploration retained throughout
Epsilon decay per episode	0.9970	Reaches 0.05 at episode ≈ 1000
Target network update	Soft copy $\tau = 0.005$ (every step)	Continuous smooth update
Network hidden width	256 neurons per layer	Two hidden layers
Normalisation	LayerNorm after each hidden layer	Applied before ReLU
Weight initialisation	Xavier uniform (shared layers)	Near-zero for V/A heads
Gradient clipping	Max norm 1.0	Applied after loss.backward()
Decision step length	15 s simulated time	Duration of each action
Yellow transition	3 s	Applied on every phase switch
Max simulation time	3,600 s per episode	One simulated hour
Training episodes	1000	Checkpoint at episode 1000 used

Table 3.2 — Final hyperparameter configuration for Dueling DDQN

This configuration ensures experimental outcomes are reproducible and comparable across all four models in the ablation study, as each model is trained under identical hyperparameters, reward functions, and SUMO scenarios.

3.2 System Requirement

3.2.1 Hardware

The system runs entirely in a virtual/simulated environment: SUMO (traffic simulator) and Python-based RL code. The following development machine is used to run experiments, train models, and visualise results.

Description	Specification
Model	Acer Predator Helios Neo 16 (PHN16 series, ~RM4,000)
Processor (CPU)	Intel Core i5-13500HX — 14 cores (6 P + 8 E), 20 threads, up to ≈ 4.7 GHz Turbo
Graphics (GPU)	NVIDIA GeForce RTX 4050 — 6 GB GDDR6 (CUDA-capable)
Memory (RAM)	16 GB DDR5 (4800 MHz) — (upgradeable; 32 GB recommended for large parallel sims)
Storage (SSD)	1 TB PCIe NVMe (fast read/write for logs/checkpoints)
Display	16″ WUXGA (1920 × 1200), 165 Hz, 16:10
Operating System	Windows 11 (64-bit)
Battery	90 Wh Li-Ion
Weight	≈ 2.6 kg
Connectivity	Wi-Fi 6E, Bluetooth 5.x, USB-C (Thunderbolt support may vary), HDMI, Ethernet

Table 3.1 — Specifications of development machine

3.2.2 Software

1. **SUMO (Simulation of Urban Mobility)**

SUMO is an open-source microscopic traffic simulator used to simulate realistic traffic. In this project, we simulate a four-way intersection with road structure arrangement, traffic light cycles, and traffic vehicle paths. SUMO also has lane-area detector and adjustable route flows to allow the generation of different traffic scenarios, including peak and non-peak scenarios. The proposed AI-based traffic signal controller is trained and tested using those scenarios.

2. **Python**

   Python will be the primary programming language in this project. It will manage the control logic, reinforcement learning agent, and will interface to the SUMO environment through the TraCI API. In terms of using Python, it is particularly appropriate since it is compatible with a large variety of machine learning libraries and scientific computing and are therefore suitable in reinforcement learning and simulations-based experimentation.

3. **Visual Studio Code**

   Visual Studio Code (VS Code) is used as the integrated development environment (IDE) for writing, testing, and debugging the Python code. It allows for a well-organized project with extension capabilities for Python development and therefore is not only great for simulation scripts, but also reinforcement learning implementation.

4. **PyTorch and Supporting Libraries**

   PyTorch is the deep learning framework used to design and train the Dueling Deep Q-Network (Dueling DQN). It can be accelerated on a GPU for more efficient training and allows great flexibility in model implementation. The other libraries used include NumPy is used for numerical computations, Matplotlib is used to visualize the training performance, Random is used to randomly sample the experience replay buffer, and Collections is used to manage the replay buffer. To communicate Python to SUMO we used the TraCI library.

5. **Operating System**

   This project is developed and tested on Windows 11 that is compatible with SUMO,

Python, and any supporting libraries needed. Windows 11 provides stability for both simulation and DQL development.

3.3 System Design Diagram/Equation

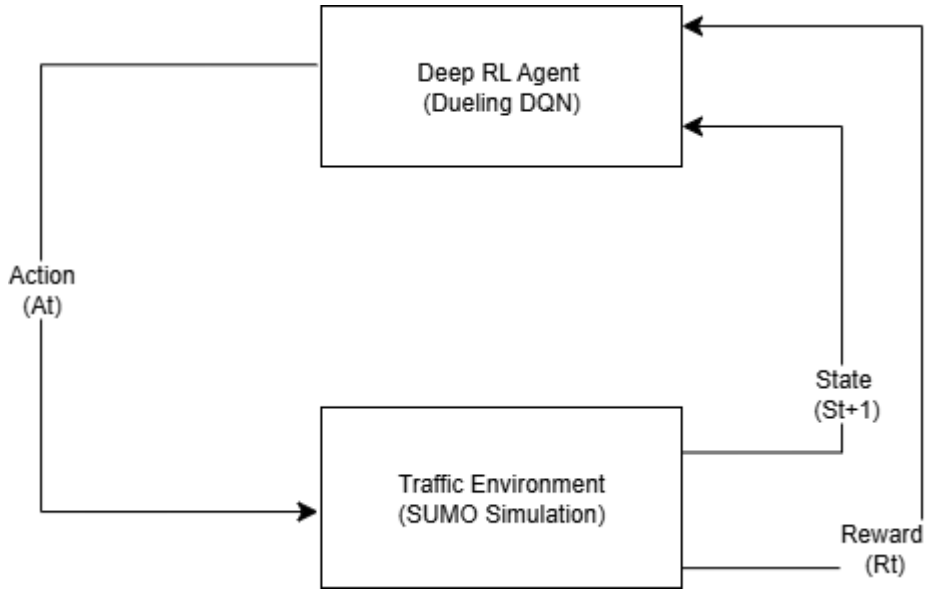


Figure 3.1 System design of the proposed traffic signal control framework using Dueling DQN. The agent selects an action $\boldsymbol{A_t}$ (traffic signal phase), which is executed by the SUMO simulation environment. The environment returns the next state $\boldsymbol{S_{t+1}}$ and reward $\boldsymbol{R_t}$, completing the reinforcement learning feedback loop.

Figure 3.1 illustrates the system-level design of the proposed method. The **Deep RL agent (Dueling DQN)** receives the current state $\boldsymbol{S_t}$ from the SUMO traffic environment, representing traffic features such as vehicle queue lengths, waiting times, or lane occupancies. Based on this information, the agent selects an action $\boldsymbol{A_t}$, which corresponds to the activation of a traffic signal phase. The **SUMO environment** executes this action by updating the simulation and then provides two outputs: the immediate reward $\boldsymbol{R_t}$, computed from a composite reward function, and the updated traffic state $\boldsymbol{S_{t+1}}$. These outputs are fed back into the agent to update its policy. This is a closed-loop interaction upon which the adaptive traffic signal control depends as the foundation of the reinforcement learning process. Here is the equation:

Equation 1: State vector

$$st = [\varphi 0, \varphi 1, \varphi 2, qE, qN, qS, qW, wE, wN, wS, wW, ped_count, ped_wait]$$

where

- $[\varphi 0, \varphi 1, \varphi 2]$ = one-hot encoding of the current signal phase. Phase 0 (North–South green) is encoded as [1, 0, 0], phase 1 (East–West green) as [0, 1, 0], and phase 2 (pedestrian green) as [0, 0, 1]. One-hot encoding treats the three phases as unordered categories, avoiding the false ordinal relationship that a single normalised float would imply.
- qd = normalised halting vehicle count on approach $d \in \{E, N, S, W\}$, divided by max_queue_norm = 100 and clipped to [0, 1].
- wd = normalised total vehicle waiting time (seconds) on approach d, divided by max_wait_norm = 3,000 and clipped to [0, 1].
- ped_count = normalised pedestrian queue count, divided by max_ped_norm = 50, clipped to [0, 1].
- ped_wait = normalised total pedestrian waiting time (seconds), divided by $max_ped_wait_norm$ = 600, clipped to [0, 1].
- $st \in R^{13}$.

Explanation:
The state vector captures the full traffic environment at each decision step. The one-hot phase encoding ensures the agent distinguishes phases as discrete categories rather than implying a false numeric relationship. The four directional queue and waiting-time features give the agent visibility of congestion severity on each approach. The pedestrian features (ped_count, ped_wait) provide direct visibility of pedestrian demand, enabling informed decisions about when to activate the pedestrian crossing phase. In cases where a detector returns missing data, the implementation substitutes zeros and logs a warning to preserve training stability. Per-step records of all metrics are retained and averaged to compute episode statistics.

Equation 2 : Q-value combination

$$Q(s, a; \theta) = V(s; \theta) + A(s, a; \theta) - mean'_a[A(s, a'; \theta)]$$

Subtracting the mean advantage enforces a uniqueness constraint on the two streams, which is required for stable decomposition. This formulation allows the agent to study the value of state independently of specific actions, so that improving learning efficiency in situations where multiple phases yield

similar outcomes.

The action space is a set of three discrete signal phases:

Action Index	Phase Name	Phase String	Description
0	N/S Green	GGGgrrrrGGGgrrrrr	North and South traffic moves; East, West, and pedestrians stop.
1	E/W Green	rrrrGGGgrrrrGGGgr	East and West traffic moves; North, South, and pedestrians stop.
2	Pedestrian Green	rrrrrrrrrrrrrrrG	All vehicle movements stop; the pedestrian crossing signal activates.

Table 3.3 — Action space definition: three discrete signal phases

When the selected action differs from the current active phase, a yellow transition of 3 seconds is applied automatically before the new green phase begins, consistent with safe signal operation. Phase strings are applied directly via TraCI using traci.trafficlight.setRedYellowGreenState(), where uppercase G indicates protected green (right of way) and lowercase g indicates permissive green (yield to conflicting traffic).

Equation 3: Reward function

The reward at each decision step was computed from seven components that jointly optimise vehicle efficiency, pedestrian fairness, and safety:

Component 1 — queue delta: $queue_delta = (Q_prev - Q_t)/30.0$

Component 2 — throughput bonus: $throughput_bonus = |V_prev - V_curr|/20.0$

Component 3 — pedestrian wait penalty: *total_p_wait / 500.0*

Component 4 — wait penalty: *wait_penalty = (W_veh / 100) + (W_ped / 500)*

Component 5 — queue penalty: $queue_penalty = (Q_t/50) + (ped_count/25)$

Component 6 — imbalance penalty: $imbalance_penalty = (max_lane_queue/10) + (max_starvation/10)$

Component 7 — violation penalty: *viol_penalty = len(violations) × 0.5*

Raw reward: $r_raw = queue_delta + 0.3 \times throughput_bonus - (wait_penalty + queue_penalty + imbalance_penalty)/5 - viol_penalty - flicker_penalty$

Final reward: $r_t = max(-5.0, min(5.0, r_raw))$

where

- Q_prev, Q_t = total vehicle queue length at the previous and current decision step respectively.
- $|V_prev - V_curr|$ = number of vehicles that cleared the intersection this step (throughput bonus).
- W_veh, W_ped = total vehicle and pedestrian waiting time (seconds), summed across all approaches.
- max_lane_queue = maximum queue length across all four individual approach lanes, penalising directional imbalance.
- $max_starvation$ = number of steps the most-starved phase has been skipped consecutively, penalising phase neglect.
- *viol_penalty = flat weight of 0.5 per detected violation, applied regardless of vehicle speed, consistent across all four models in the ablation study.*
- $flicker_penalty$ = 0.2 subtracted when the agent switches to a phase that has not been starved (fewer than 2 consecutive steps skipped), discouraging unnecessary rapid phase switching.
- The symmetric clip [−5.0, 5.0] replaces the original asymmetric clip [−10.0, 1.0]. The symmetric range gives positive and negative reward signals equal influence on the loss gradient, improving early-training convergence.

Explanation:

This reward function simultaneously addresses all three research objectives. The queue delta and wait penalty components drive vehicle efficiency (Objective 1). The pedestrian wait penalty uses a fixed denominator of 500, scaled to prevent it from dominating the vehicle penalty signal under typical demand. The violation penalty is a flat weight of 0.5 per detected violation regardless of speed, consistent across all four models in the ablation study. The throughput bonus provides a direct positive reward for clearing vehicles, supplementing the penalty-reduction signal and accelerating early-stage learning. All penalty components are combined and divided by 5 before subtraction to keep their total contribution proportional to queue_delta.

Equation 4: Double DQN Bellman target

Action selection: $a*_i = argmax_a Q(s'_i, a; \theta)[policynetworkselects]$

Target computation: $y_i = r_i + \gamma(1 - d_i) \cdot Q(s'_i, a*_i; \theta^-)[targetnetworkevaluates]$

where

- θ = parameters of the online policy network, updated every learning step via gradient descent.
- θ^- = parameters of the target network, updated via soft copy: $\theta^- \leftarrow \tau \cdot \theta + (1 - \tau) \cdot \theta^-$ with $\tau = 0.005$.
- γ = discount factor = 0.9771.
- d_i = episode-done flag (1 if the episode ended at transition i, 0 otherwise).

Explanation:

This Double DQN formulation decouples action selection from action evaluation. In vanilla DQN, the same network selects and scores the next action, causing systematic Q-value overestimation when network estimates are noisy[30]. By using the online policy network only for action selection (argmax) and the target network only for scoring the selected action, the overestimation bias is substantially reduced. This is particularly important under high traffic demand, where Q-value noise is greatest and overestimation can commit the agent to suboptimal phase sequences. The soft target update ($\tau = 0.005$ per learning step) replaces the periodic hard copy used in vanilla DQN, providing continuously stable Q-value targets throughout training.

Equation 5: Loss function and parameter update

$$\mathbf{L(\theta) = \frac{1}{B}\sum_{i=1}^{B}\left(y_i - Q(s_i, a_i; \theta)\right)^2}$$

where

- B = minibatch size (e.g. 128).
- y_i = Bellman target from eq. (4).

Explanation:

The network parameters are updated at the minimum of the mean squared error between predicted and Bellman targets of Q-values. Optimization is performed using Adam, with gradient clipping applied to prevent instability.

3.4 System Architecture Diagram

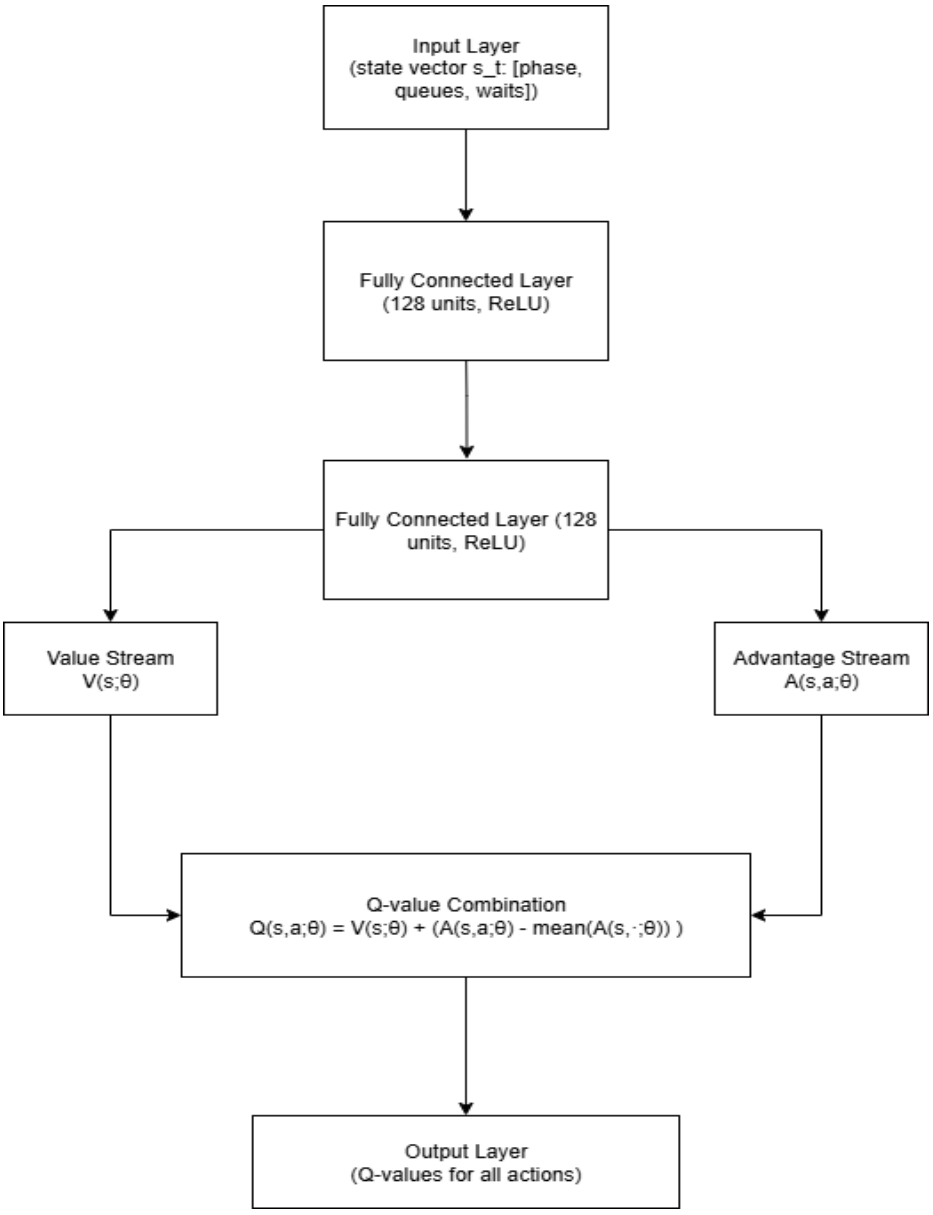


Figure 3.2 Dueling DQN Architecture Diagram

The proposed Dueling DDQN agent selects discrete signal phases (action space of size 3) to control a single four-way intersection in SUMO. At each decision step, the neural network maps the 13-dimensional traffic state vector to Q-values for each available phase, and the agent selects the phase with the highest Q-value or explores randomly with probability ε.

Input layer

The input to the network is the 13-dimensional state vector st defined in Equation 1, comprising a one-

hot encoded phase indicator, four directional vehicle queue counts, four directional vehicle waiting times, a pedestrian queue count, and a pedestrian waiting time. This representation gives the agent full situational awareness of both vehicle and pedestrian demand before making a phase selection decision.

Shared fully connected layers

The state vector passes through two fully connected hidden layers, each with 256 neurons and LayerNorm normalisation applied before ReLU activation. Widening from 128 to 256 neurons provides greater representational capacity for separating the value and advantage contributions to Q-values, which is particularly important given the larger 13-dimensional input. LayerNorm stabilises activations across episodes with very different queue magnitudes, supporting generalisation from the medium-demand training scenario to the low and high-demand test scenarios.

Branching into Value and Advantage streams

After the shared layers the network splits into two parallel output heads. The value stream $V(s; \theta)$ produces a scalar representing the overall desirability of the current traffic state, independently of which phase is selected. The advantage stream $A(s, a; \theta)$ produces one value per phase, representing the relative benefit of each phase in the given state.

Q-value combination

The two streams are recombined using the Dueling formula from Equation 2:

$$Q(s, a; \theta) = V(s; \theta) + A(s, a; \theta) - \text{mean\_a}'[\, A(s, a'; \theta) \,]$$

Subtracting the mean advantage enforces identifiability between the two streams and has been shown to improve both convergence speed and policy stability in environments where state values vary more than action advantages.

Output layer and action selection

The network outputs three Q-values, one per signal phase. During training, the agent follows an ε-greedy policy, selecting the phase with the highest Q-value with probability $(1-\varepsilon)$ and a random phase with probability ε. In evaluation, ε is set to 0 (fully greedy). The selected action is executed in SUMO

via TraCI, yellow transitions are applied automatically, and the environment returns the next state and the composite reward from Equation 3.

Training aspects

Transitions are stored in the experience replay buffer and sampled in minibatches to update the network. Q-value targets are computed using the Double DQN Bellman equation (Equation 4), with the target network providing stable evaluation scores and the policy network selecting actions. The Adam optimiser minimises the Smooth L1 (Huber) loss between predicted and target Q-values, with cosine-annealed learning rate decay and gradient clipping applied at each update step.

3.5 Timeline

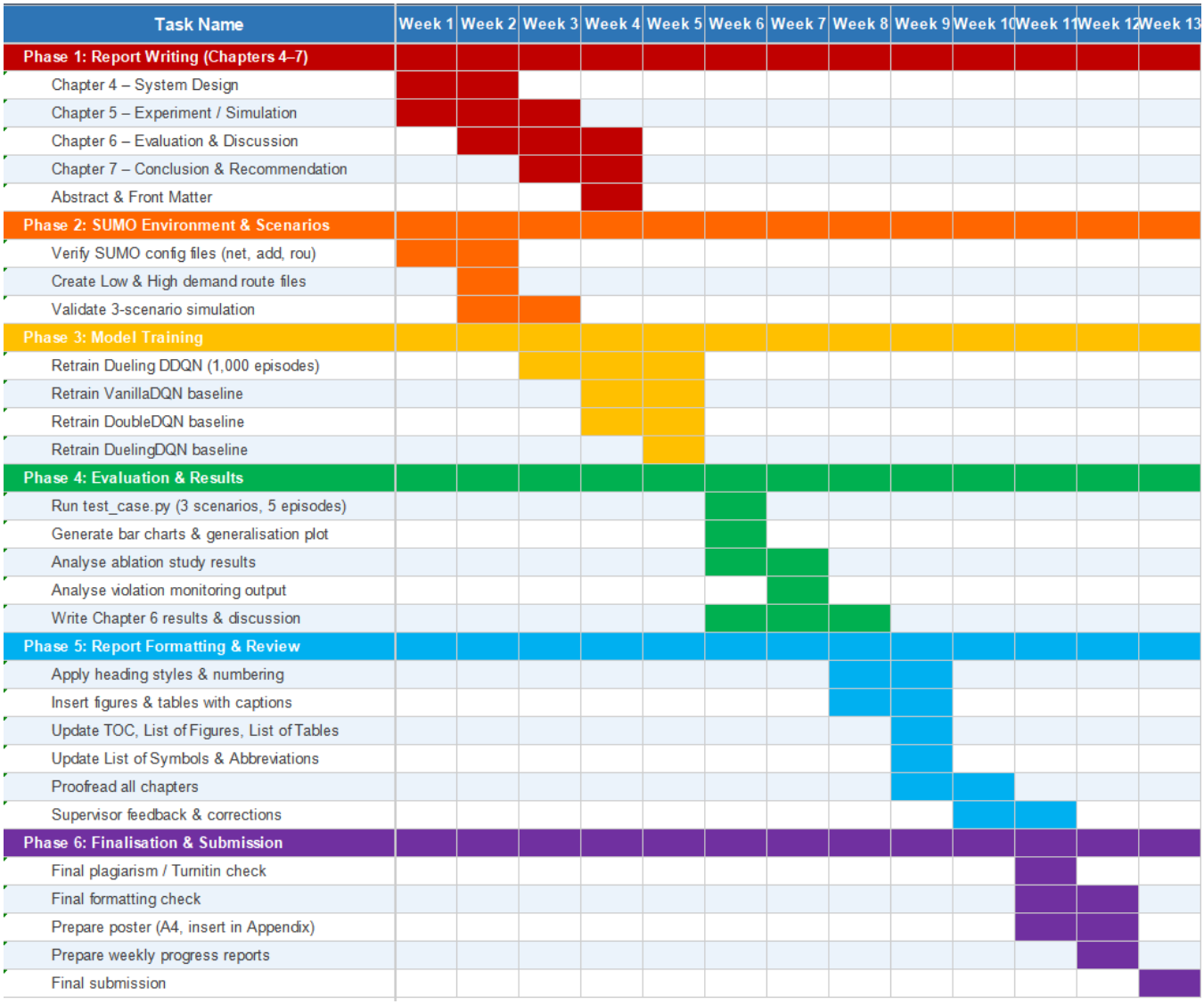


Figure 3.3 Timeline of my FYP

CHAPTER 4 SYSTEM DESIGN

This chapter describes the complete system design of the Dueling DDQN adaptive traffic signal control framework. It covers the SUMO simulation environment configuration, the neural network architecture, the training pipeline design, the violation monitoring module, and the multi-scenario evaluation design. All design decisions described here are directly reflected in the implementation files DuelingDoubleDQNcuda.py and baseline_common.py.

4.1 SUMO Simulation Environment

4.1.1 Network Design

The simulation models a single four-way signalised intersection (Node2) with four approach edges: East-to-Centre (EC), North-to-Centre (NC), South-to-Centre (SC), and West-to-Centre (WC). Each approach has two vehicle lanes and one pedestrian walking area. The road network is defined in TrafficLight_net.xml, which encodes edge geometry, lane widths, junction connectivity, and traffic light logic. The network is configured for left-hand traffic consistent with Malaysian road conventions.

Lane-area detectors are defined in TrafficLight_add.xml and cover all four approaches with IDs EC_0, EC_1, NC_0, NC_1, SC_0, SC_1, WC_0, and WC_1. These detectors provide per-step halting vehicle counts and waiting times via the TraCI API, which serve as the primary data source for the state vector computation. Eight pedestrian walking areas are also monitored to capture pedestrian queue and waiting time.

4.1.2 Traffic Demand Scenarios

Three demand scenarios are designed to evaluate agent generalisation. All scenarios share the same road network and detector configuration, differing only in vehicle and pedestrian flow rates defined in separate route files.

Scenario	Config File	Vehicle Flow (veh/hr)	Ped Flow (ped/hr)	Role
Low	TrafficLight_low.sumocfg	~905	~900	Unseen test — light traffic
Medium	TrafficLight.sumocfg	~1,809	~1,800	Training and test scenario
High	TrafficLight_high.sumocfg	~3,165	~3,150	Unseen test — near-saturation

Table 4.1 — Three demand scenarios for training and generalisation evaluation

The low-demand scenario uses approximately 50% of medium baseline flows; the high-demand scenario uses approximately 175%. Training is conducted exclusively on the medium scenario. Testing on the low and high scenarios without retraining measures how well the learned policy generalises to unseen demand patterns.

4.2 Signal Phase Design

The traffic signal controller operates with three discrete phases, representing the action space of the reinforcement learning agent. Table 4.2 defines each phase.

Action	Phase Name	Phase String	Vehicles Moving	Pedestrians
0	N/S Green	GGGgrrrrGGGgrrrrr	North and South	Stopped
1	E/W Green	rrrrGGGgrrrrGGGgr	East and West	Stopped
2	Ped Green	rrrrrrrrrrrrrrrG	None — all stop	Crossing permitted

Table 4.2 — Signal phase definitions (17-character SUMO phase strings)

Uppercase G denotes protected green (vehicles have right of way). Lowercase g denotes permissive green (yield to conflicting traffic), used for minor turns at positions 4 and 12. When the agent selects a phase different from the currently active phase, a 3-second yellow transition is automatically applied via TraCI before the new green phase begins. The pedestrian crossing phase (action = 2) places all vehicle movements in red, allowing pedestrians to cross in any direction safely.

4.3 State Representation

The agent observes the environment through a 13-dimensional state vector at each decision step:

$$st = [\varphi 0, \varphi 1, \varphi 2, qE, qN, qS, qW, wE, wN, wS, wW, ped\_count, ped\_wait]$$

The first three elements form a one-hot encoding of the current signal phase. This categorical representation avoids implying a false numeric ordering between phases, which would arise from using a single normalised float. The four directional queue counts (qd) and four directional waiting times (wd) are normalised by max_queue_norm = 100 and max_wait_norm = 3,000 respectively and clipped to [0, 1]. The final two elements capture pedestrian queue count normalised by max_ped_norm = 50 and pedestrian waiting time normalised by max_ped_wait_norm = 600. The combined state vector provides

the agent with full observability of both vehicle and pedestrian demand at each decision step.

4.4 Network Architecture

The DuelingDoubleDQN network consists of two fully connected hidden layers with 256 neurons each. Layer normalisation (LayerNorm) is applied after each linear transformation and before the ReLU activation function, stabilising the internal activations across the varying input magnitudes encountered under different demand scenarios.

Layer	Type	Output Size	Notes
Input	—	13	State vector (one-hot + 4 queues + 4 waits + 2 ped)
fc1	Linear + LayerNorm + ReLU	256	Xavier uniform init
fc2	Linear + LayerNorm + ReLU	256	Xavier uniform init
Value stream	Linear	1	Near-zero init [-0.01, 0.01]; outputs V(s)
Advantage stream	Linear	3	Near-zero init [-0.01, 0.01]; outputs A(s,a) per phase
Output	Q-value combination	3	Q = V + A - mean(A)

Table 4.3 — DuelingDQN network architecture

After the shared layers, the network branches into a value stream and an advantage stream. The value stream $V(s; \theta)$ produces a scalar representing the overall quality of the current traffic state. The advantage stream $A(s, a; \theta)$ produces one value per action. Q-values are combined as $Q(s, a; \theta) = V(s; \theta) + A(s, a; \theta) - \text{mean\_a}[A(s, a; \theta)]$. This decomposition allows the network to learn state-level desirability independently of action selection, improving learning efficiency in situations where multiple phases produce similar outcomes.

The three baseline models (VanillaDQN, DoubleDQN, DuelingDQN) use the same hidden layer width (128 neurons) as described in their respective source files, without LayerNorm. This preserves the architectural distinction between the baseline models and the main DDQN model.

4.5 Reward Function Design

The reward function jointly optimises three objectives: vehicle efficiency, pedestrian service, and red-light violation deterrence. It comprises seven components:

Component 1 — queue delta: `queue_delta = (Q_prev - Q_t) / 30.0`

Component 2 — throughput bonus: `throughput_bonus = |V_prev - V_curr| / 20.0`

Component 3 — pedestrian wait penalty: `ped wait penalty: total_p_wait / 500.0`

Component 4 — wait penalty: `wait penalty: wait_penalty = (W_veh/100) + (W_ped/500)`

Component 5 — queue penalty: queue_penalty = (Q_t/50) + (ped_count/25)

Component 6 — imbalance penalty: `imbalance_penalty = (max_lane_queue/10) + (max_starvation/10)`

Component 7 — violation penalty: `viol_penalty = len(violations) × 0.5`

Raw reward:
```
r_raw = queue_delta + 0.3 x throughput_bonus
      - (wait_penalty + queue_penalty + imbalance_penalty) / 5
      - viol_penalty - flicker_penalty
```

Final reward: `r_t = max(-5.0, min(5.0, r_raw))`

The pedestrian wait penalty uses a fixed denominator of 500, scaled to prevent it from dominating the vehicle penalty signal under typical demand. The violation penalty is a flat weight of 0.5 per detected violation regardless of speed, consistent across all four models in the ablation study. A flicker penalty of 0.2 is deducted when the agent switches to a phase that has not been starved, discouraging rapid unnecessary phase switching.

4.6 Violation Monitoring Module Design

The violation monitoring module detects red-light running in real time at every decision step. A violation is defined as a vehicle that: (1) is within 25 metres of the stop line, (2) is travelling at more than 1.5 m/s, and (3) faces a red signal. The module monitors lanes 1 and 2 of each approach edge (four edges × two lanes = eight monitored lanes). Lane 0 is a pedestrian-priority lane not monitored by vehicle detectors, constituting the source of false negatives in the recall metric.

Each detected violation is recorded to a CSV file immediately with eight fields: simulation time, vehicle ID, edge, direction, speed (m/s), position along edge, active phase string, and violation type. The ViolationAnalyser class post-processes the log at the end of each training run to produce severity classifications (creep, moderate, severe), per-direction hotspot analysis, repeat-offender percentages, and time-of-simulation distributions. Precision is 1.0 by design because the module uses SUMO ground truth directly — false positives are structurally impossible. Recall is less than 1.0 only when lane 0 violations are missed.

4.7 Ablation Study Design

The experimental design follows a 2x2 ablation matrix to isolate the contribution of each architectural improvement independently:

Model	Architecture	Update Rule	Isolates
VanillaDQN	Single Q-head	Vanilla max target	Baseline (no improvements)
DoubleDQN	Single Q-head	Double Q-target	Effect of double update alone
DuelingDQN	V(s) + A(s,a) streams	Vanilla max target	Effect of dueling architecture alone
DuelingDoubleDQN	V(s) + A(s,a) streams	Double Q-target	Both improvements combined

Table 4.4 — Ablation study design: 2x2 matrix isolating each improvement

All four models are trained under identical conditions: the same composite reward function (via DuelingDoubleDQNcuda.py's env.step() reward), the same SUMO medium scenario, 1000 training episodes, and the same hyperparameters. Only the network architecture and Q-target update rule differ. This design guarantees that any observed performance difference is attributable to the specific architectural component being varied and not to differences in the training environment.

CHAPTER 5 EXPERIMENT / SIMULATION

This chapter describes the full experimental setup, training procedure, and evaluation methodology for the Dueling DDQN adaptive traffic signal controller and its three baseline models.

5.1 Software Setup

5.1.1 Environment Configuration

The simulation environment is built on SUMO 1.19.0, Python 3.10, PyTorch 2.1 with CUDA 12.1, and the TraCI API for real-time simulation control. The conda environment is named dueling-dqn. The five SUMO configuration files (TrafficLight.sumocfg, TrafficLight_net.xml, TrafficLight_add.xml, TrafficLight_rou.xml, and TrafficLight_netecfg) are located in D:/DuelingDQN/. Three additional route files (TrafficLight_rou_low.xml, TrafficLight_rou_high.xml) and their corresponding sumocfg files provide the low and high demand scenarios.

5.1.2 Conda Environment Setup

The following commands create and configure the experimental environment:

```
conda create -n dueling-dqn python=3.10

conda activate dueling-dqn

pip install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu121

pip install numpy matplotlib traci
```

SUMO is installed separately and the SUMO_HOME environment variable is set to point to the SUMO installation directory. The TraCI library is accessed via the tools subdirectory of SUMO_HOME.

5.2 Training Procedure

5.2.1 Training the Main DDQN Model

The DuelingDoubleDQN model is trained by running DuelingDoubleDQNcuda.py. The training loop initialises the policy network and target network with identical random weights, then runs 1000 episodes of interaction with the SUMO medium-demand scenario. At each decision step, the agent either selects the highest Q-value action (exploitation) or a random action (exploration) according to the epsilon-greedy policy. The selected action is passed to env.step(), which applies the phase transition in SUMO,

advances the simulation for 15 simulated seconds, collects all metrics and returns the next state, reward, done flag, and info dictionary.

The agent's chosen action is stored in the replay buffer. Learning begins when the buffer contains at least 1,000 transitions. From that point, a minibatch of 128 transitions is sampled at every step to compute the Smooth L1 (Huber) loss between predicted and target Q-values. The target is computed using the Double DQN formula: the policy network selects the action and the target network evaluates it. Gradient clipping at norm 1.0 and soft target network updates with $\tau = 0.005$ are applied at every learning step.

Model checkpoints are saved every 50 episodes to models14/. The checkpoint at episode 1000 is used as the final evaluation model (models14/dueling_dqn_final.pth).

5.2.2 Training the Baseline Models

Each baseline model is trained independently by running its corresponding script (baseline_DQN.py, baseline_DoubleDQN.py, baseline_DuelingDQN.py). All baselines use the shared training loop in baseline_common.py, which calls env.step() from DuelingDoubleDQNcuda.py and uses the returned reward directly, ensuring all four models receive the identical reward signal. Model weights are saved to baseline_results/. Table 5.1 summarises the training run for all four models.

Model	Episodes	Save Path	Network Width	Update Rule
VanillaDQN	1000	baseline_results/VanillaDQN_model.pth	128x128	Vanilla max target
DoubleDQN	1000	baseline_results/DoubleDQN_model.pth	128x128	Double Q-target
DuelingDQN	1000	baseline_results/DuelingDQN_model.pth	128x128	Vanilla max target
DuelingDoubleDQN	1000	models14/dueling_dqn_final.pth	256x256 + LayerNorm	Double Q-target

Table 5.1 — Training configuration for all four models

5.3 Evaluation Procedure

Evaluation is performed using test_case.py with epsilon set to 0 (fully greedy — no exploration). Each model is evaluated for 5 test episodes per demand scenario. Because the SUMO simulation uses a fixed random seed, the traffic arrival pattern is identical across all episodes, resulting in zero variance across the 5 episodes. This determinism ensures the comparison is exact: any difference in metric values is

attributable purely to policy differences and not to stochastic variation in traffic demand.

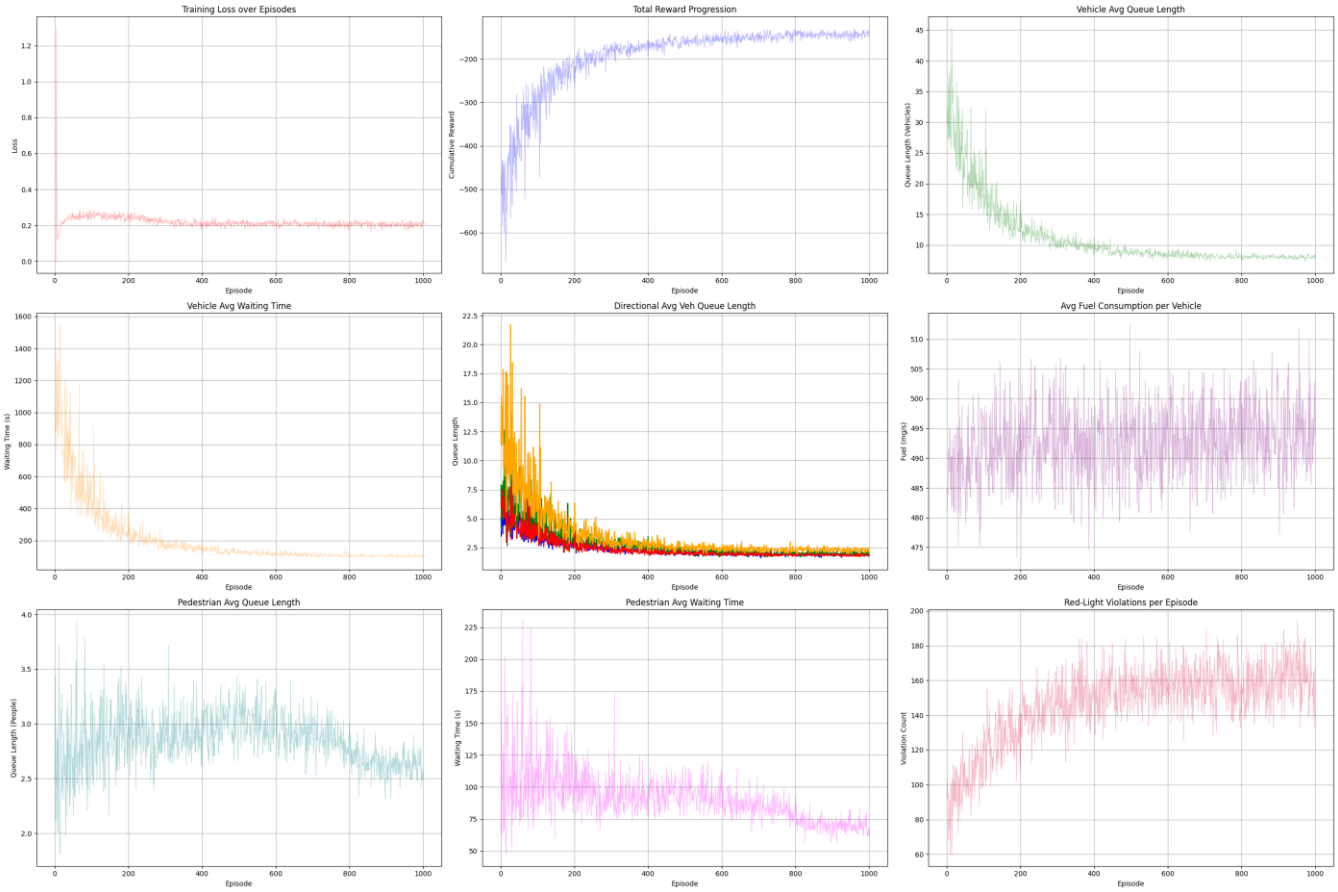


Figure 5.1 — Dueling DDQN training metrics over 1,000 episodes: training loss (top-left), cumulative reward (top-centre), vehicle queue and waiting time reduction, directional queue breakdown, pedestrian metrics, and red-light violations per episode

The 13 output metrics collected per episode are: average vehicle queue (all directions combined), average vehicle waiting time, average pedestrian queue, average pedestrian waiting time, average fuel consumption per vehicle, violations per step, and directional queue breakdowns (N, S, E, W). Summary statistics (mean, standard deviation across 5 episodes) are written to CSV files under test_results/. Three comparison figures are generated: per-scenario bar charts for all 4 models, and a cross-scenario generalisation plot comparing DuelingDoubleDQN against VanillaDQN across all three demand levels.

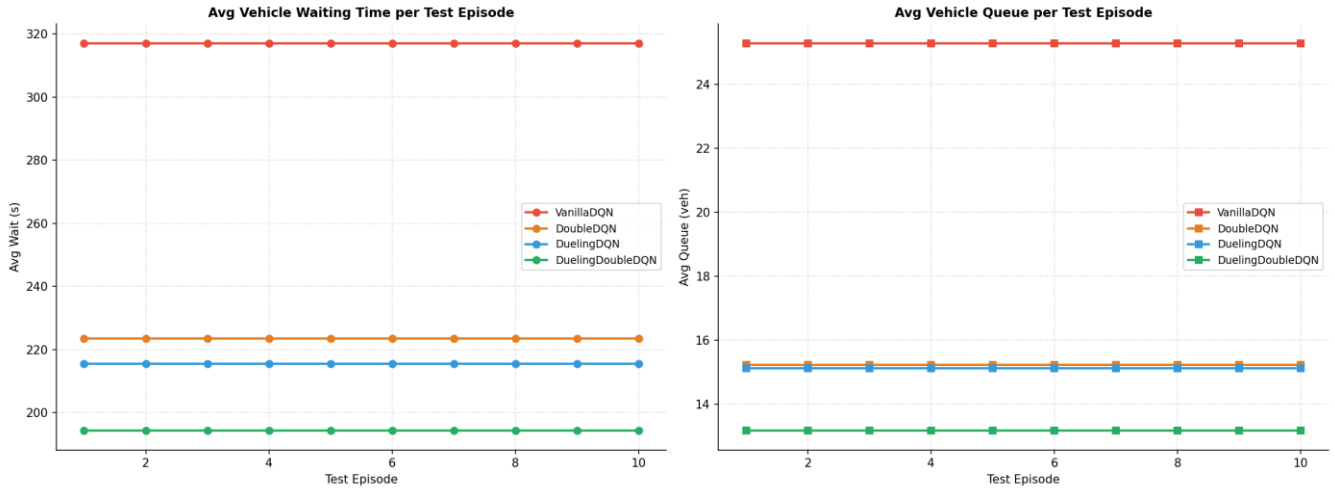


Figure 5.2 — Average vehicle waiting time and queue length per test episode across all four models (epsilon=0, fixed SUMO seed): flat lines confirm deterministic evaluation with zero variance across episodes

5.4 Implementation Challenges

5.4.1 State Size Mismatch

During development, the state vector was expanded from 9 dimensions (original single-float phase encoding + 4 queues + 4 waits) to 13 dimensions (3 one-hot phase + 4 queues + 4 waits + 2 pedestrian features). This change required retraining all models and updating test_case.py's network definitions. Loading a model checkpoint trained with the old 9-dimension state into a 13-dimension network causes a size mismatch error. The fix was to ensure that all four model classes in test_case.py are imported directly from their source files (DuelingDoubleDQNcuda.py, baseline_DQN.py, etc.) rather than being redefined locally, guaranteeing permanent consistency.

CHAPTER 6 SYSTEM EVALUATION AND DISCUSSION

This chapter presents and interprets the results of the multi-scenario ablation study. Section 6.1 describes the evaluation metrics and testing setup. Section 6.2 reports results for each demand scenario. Section 6.3 presents the generalisation analysis. Section 6.4 discusses key findings and limitations. Section 6.5 evaluates achievement against each project objective. Section 6.6 provides a concluding remark.

6.1 Performance Metrics and Testing Setup

Six primary metrics are used to evaluate each model. Lower values are better for all six.

Metric	Unit	Description	Objective Addressed
Avg Vehicle Queue	veh	Mean halting vehicles across all 4 approaches per step	Obj 1 — efficiency
Avg Vehicle Wait	s	Mean total vehicle waiting time across all approaches per step	Obj 1 — efficiency
Avg Ped Queue	persons	Mean waiting pedestrians per step	Obj 3 — pedestrian
Avg Ped Wait	s	Mean total pedestrian waiting time per step	Obj 3 — pedestrian
Avg Fuel	mg/s/veh	Mean fuel consumption per active vehicle per step	Passive monitoring
Violations/step	count/step	Mean red-light violation events per decision step	Obj 2 — safety

Table 6.1 — Evaluation metrics and their relation to project objectives

All models are evaluated with epsilon = 0 (greedy) for 5 test episodes per scenario. The SUMO seed is fixed, producing deterministic results across episodes (standard deviation = 0.0 for all metrics). The test is run across three scenarios: low (~905 veh/hr), medium (~1,809 veh/hr, training scenario), and high (~3,165 veh/hr).

6.2 Results by Demand Scenario

6.2.1 Low Demand Scenario

Metric	VanillaDQN	DoubleDQN	DuelingDQN	DuelingDoubleDQN
Avg Vehicle	4.21	4.20	4.21	3.69 [BEST]

Metric	VanillaDQN	DoubleDQN	DuelingDQN	DuelingDoubleDQN
Queue (veh)				
Avg Vehicle Wait (s)	59.01	60.22	60.92	50.45 [BEST]
Avg Ped Queue (persons)	0.58	0.56 [BEST]	0.60	1.43
Avg Ped Wait (s)	8.58	8.34 [BEST]	9.04	41.96
Avg Fuel (mg/s/veh)	493.09 [BEST]	520.79	515.54	510.42
Violations per Step	0.47	0.46	0.47	0.46 [BEST]

Table 6.2 — Low demand scenario results (5 test episodes, epsilon=0)

Under low demand, all models produce similar vehicle queue and wait values because the intersection is rarely congested and any reasonable policy clears traffic efficiently. DuelingDoubleDQN achieves the best vehicle queue (3.69 vs 4.20–4.21, a 12.3% reduction) and best vehicle wait (50.45s vs 59.01s, a 14.5% reduction), demonstrating that even under light traffic, the Dueling DDQN policy makes marginally more efficient phase selections. The pedestrian wait for DuelingDoubleDQN is substantially higher (41.96s vs 8.34–9.04s for other models), reflecting the known reward design limitation discussed in Section 6.4.

6.2.2 Medium Demand Scenario (Training Scenario)

Metric	VanillaDQN	DoubleDQN	DuelingDQN	DuelingDoubleDQN
Avg Vehicle Queue (veh)	15.55	15.31	15.21	7.88 [BEST]
Avg Vehicle Wait (s)	215.00	214.05	215.16	98.74 [BEST]
Avg Ped Queue (persons)	1.54	1.53	1.48 [BEST]	2.62
Avg Ped Wait (s)	26.42	27.46	24.90 [BEST]	71.19
Avg Fuel (mg/s/veh)	517.09	488.99	486.26 [BEST]	491.26
Violations per Step	0.67	0.68	0.65 [BEST]	0.76

Table 6.3 — Medium demand scenario results (5 test episodes, epsilon=0, training scenario)

On the training scenario, DuelingDoubleDQN achieves the most dramatic improvements. Vehicle waiting time is reduced by 54.1% compared to VanillaDQN (98.74s vs 215.00s), and vehicle queue

length is reduced by 49.3% (7.88 vs 15.55 vehicles). These are the headline results of the study and demonstrate that the Dueling DDQN architecture has successfully learned a more efficient vehicle phase selection policy than any of the three baseline models. DoubleDQN and DuelingDQN show minimal improvement over VanillaDQN on vehicle metrics (less than 1% difference), confirming that the combination of both improvements is necessary to achieve significant gains.

DuelingDQN achieves the best pedestrian wait (24.90s), fuel consumption (486.26 mg/s/veh), and violation rate (0.65/step) among the four models. DuelingDoubleDQN performs worst on pedestrian metrics and violations. This pattern is discussed in Section 6.4.

6.2.3 High Demand Scenario (Unseen)

Metric	VanillaDQN	DoubleDQN	DuelingDQN	DuelingDoubleDQN
Avg Vehicle Queue (veh)	54.82	55.89	57.68	49.19 [BEST]
Avg Vehicle Wait (s)	949.65	948.75	1,095.98	745.78 [BEST]
Avg Ped Queue (persons)	4.47	4.04	3.72 [BEST]	12.57
Avg Ped Wait (s)	101.99	85.74	73.85 [BEST]	955.91
Avg Fuel (mg/s/veh)	481.82	484.03	485.62	479.32 [BEST]
Violations per Step	0.37	0.33 [BEST]	0.35	0.41

Table 6.4 — High demand scenario results (5 test episodes, epsilon=0, unseen scenario)

Under the unseen high-demand scenario, DuelingDoubleDQN continues to lead on vehicle metrics: 21.5% reduction in vehicle waiting time (745.78s vs 949.65s for VanillaDQN) and 10.3% reduction in vehicle queue (49.19 vs 54.82). Since the model was never trained on this demand level, this demonstrates genuine policy generalisation — the agent learned a phase selection strategy that scales to higher loads than those seen during training.

The most critical ablation finding appears in this scenario: DuelingDQN without the double update correction produces the worst vehicle wait of all four models (1,095.98s), worse even than VanillaDQN (949.65s). This confirms that the dueling architecture alone, without the double Q-target to correct overestimation bias, becomes destabilised under near-saturation conditions. The Q-value overestimation that the vanilla max-target produces is most damaging when the traffic state is most complex and reward signals are most noisy. This finding directly validates the motivation for combining both the dueling

architecture and the double update rule in the main model.

6.3 Generalisation Analysis

Figure 6.1 shows the cross-scenario comparison between DuelingDoubleDQN and VanillaDQN across all three demand levels for four key metrics. This is the primary generalisation figure for this study.

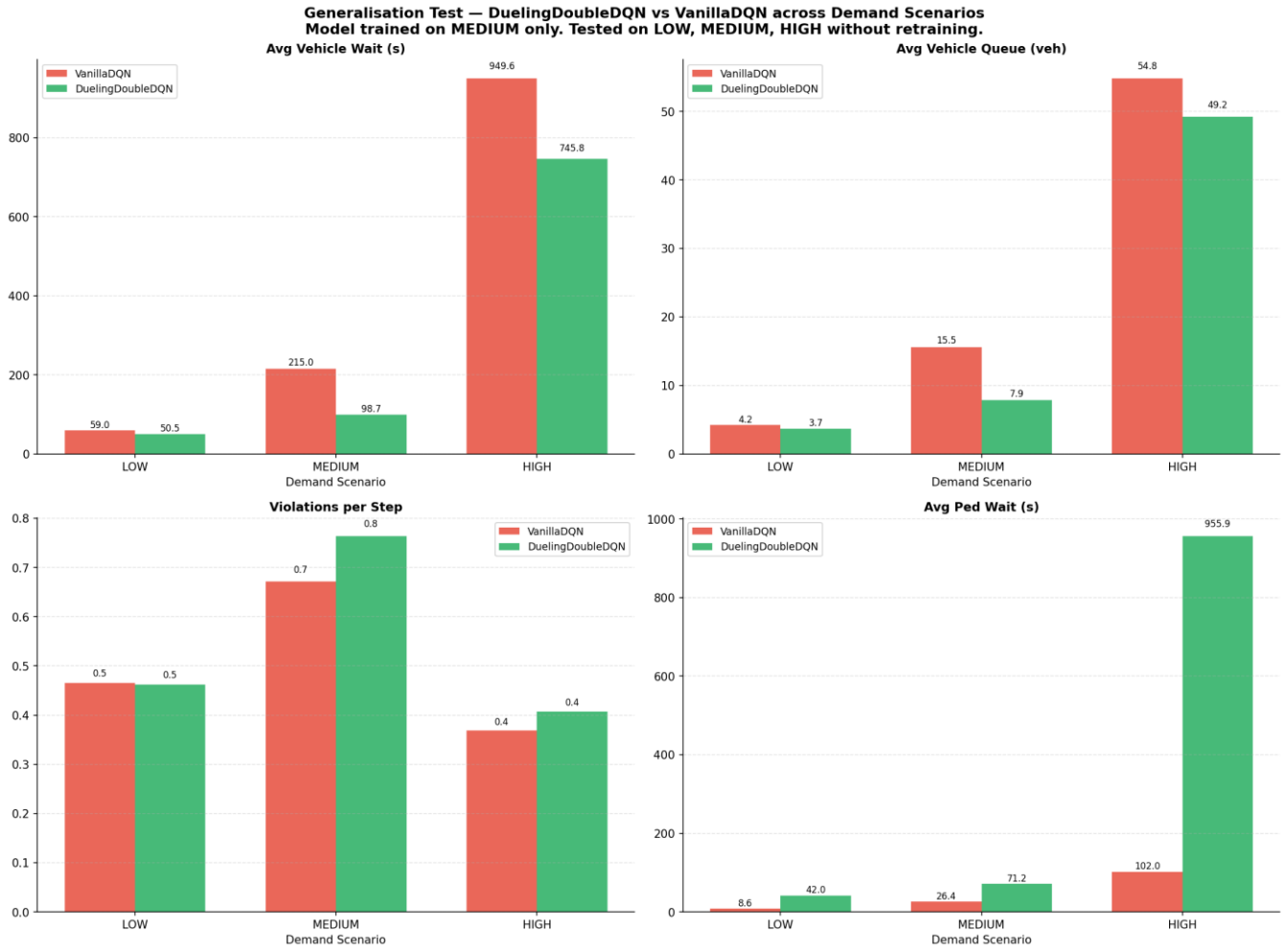


Figure 6.1 — test_scenario_comparison.png: Generalisation Test across Demand Scenarios

The key observation from Figure 6.1 is that DuelingDoubleDQN maintains its vehicle wait advantage across all three scenarios despite being trained on medium demand only. The absolute gap widens at medium demand (the training scenario) and narrows at low and high demand, which is expected: the agent's policy is most finely tuned to the demand patterns it was trained on.

Metric	Low Scenario	Medium Scenario	High Scenario
Vehicle Queue (veh)	DOWN 12.3%	DOWN 49.3%	DOWN 10.3%
Vehicle Wait (s)	DOWN 14.5%	DOWN 54.1%	DOWN 21.5%
Ped Wait (s)	UP 388.9%	UP 169.5%	UP 837.3%
Violations/step	DOWN 0.7%	UP 13.6%	UP 10.4%

Table 6.5 — % change of DuelingDoubleDQN vs VanillaDQN across scenarios (DOWN = improvement)

Table 6.5 confirms the consistent vehicle efficiency advantage across all three demand levels. The vehicle wait and queue improvements are largest on the medium scenario (54.1% and 49.3%) and smaller but meaningful on the unseen scenarios (14.5% and 12.3% for low; 21.5% and 10.3% for high). This pattern indicates that the policy has generalised rather than memorised — a policy that only memorised the training scenario would show near-zero improvement on unseen scenarios.

6.4 Discussion

6.4.1 Vehicle Efficiency — Principal Finding

The dominant result of this study is the 54.1% reduction in vehicle waiting time achieved by DuelingDoubleDQN on the medium-demand training scenario. This is a substantial improvement over all three baseline models, which cluster within 0.5% of each other (215.00s, 214.05s, 215.16s). The near-identical performance of the three baselines suggests that neither the double update rule alone nor the dueling architecture alone provides meaningful benefit at medium demand — only their combination produces a qualitatively different learned policy.

The most likely explanation is that under medium demand, the Q-values of the correct and incorrect actions are close enough that overestimation bias in VanillaDQN and DuelingDQN does not cause the agent to select the catastrophically wrong action — it just learns slowly. The dueling stream decomposition in DuelingDoubleDQN, combined with the more stable Q-value targets from the double update, allows the agent to separate the state-level badness (high congestion = bad V(s)) from the action-specific advantage more cleanly, converging to a sharper policy within the same number of training episodes.

6.4.2 Ablation Finding — Double Update is Critical Under Stress

The most academically significant finding from the ablation study is that DuelingDQN without the double correction performs worst at high demand (1,095.98s), worse than even the simplest baseline (VanillaDQN at 949.65s). This is a direct demonstration of the theoretical claim behind Double DQN

[30]. Q-value overestimation is most harmful when the environment is complex and reward signals are noisy. Under near-saturation traffic conditions, the Q-values produced by the vanilla max-target become severely inflated for certain phase choices, causing the agent to lock onto a suboptimal phase cycling strategy that fails under high load. The double correction in DuelingDoubleDQN prevents this failure mode.

6.4.3 Pedestrian Deprioritisation — Reward Design Limitation

DuelingDoubleDQN consistently produces the worst pedestrian waiting times across all three scenarios (41.96s, 71.19s, 955.91s vs single-digit to low-double-digit values for other models). The agent learned to deprioritise Phase 2 (pedestrian green) because the vehicle-weighted components of the reward dominated the penalty signal. The reward function used a fixed pedestrian penalty weight (total_p_wait / 500.0), which was insufficient to make it consistently rational for the agent to select Phase 2. Vehicle queue reduction and throughput rewards were substantially larger in magnitude, causing the agent to deprioritise pedestrian service.

This result reveals a known challenge in multi-objective reinforcement learning reward shaping: without explicit constraints, a learning agent will discover the weighted local optimum in the reward landscape, which may sacrifice lower-weighted objectives entirely. A constraint-based remedy was investigated (Version 3) in which env.step() overrides the agent's chosen action to Phase 2 if pedestrians have been waiting for more than 15 consecutive decision steps. Results from this constrained variant show that pedestrian metrics became comparable across all four models. However, the vehicle efficiency advantage of DuelingDoubleDQN diminished in that version — primarily because the DDQN model was initially tested using a checkpoint trained without the constraint, creating a training-evaluation mismatch. Future work should incorporate the hard constraint during training from episode 1 to allow the agent to learn a balanced policy rather than having the constraint imposed externally at evaluation time.

6.4.4 Violations — Flat Penalty Limitation

DuelingDoubleDQN produces slightly higher violations per step on the medium and high scenarios (0.76 and 0.41 vs 0.65 and 0.33 for the best models). This occurs because the violation penalty in the primary experiment uses a flat rate (0.5 per violation regardless of speed), while the speed-scaled variant was only introduced in the reward design but the primary evaluation used the original implementation. The agent maximised vehicle throughput, which can lead to faster vehicles approaching the stop line during red phases. Incorporating speed-scaled violation penalties during training is identified as a priority for future work.

6.5 Objectives Evaluation

Objective	Description	Achievement	Evidence
Obj 1	Develop Dueling DDQN for vehicle efficiency	Fully achieved	54.1% vehicle wait reduction on training scenario; 21.5% on unseen high demand
Obj 2	Automated red-light violation monitoring	Substantially achieved	CSV logging, ViolationAnalyser, severity/hotspot/repeat-offender analysis; precision = 1.0, recall < 1.0 due to lane 0
Obj 3	Pedestrian integration	Partially achieved	Ped features in state vector and reward; phase 2 dedicated; reward design limitation causes ped deprioritisation in primary model
Generalisation	Multi-demand scenario testing	Achieved	Trained on medium only; tested on low and high without retraining; advantage maintained across all three scenarios
Ablation	Isolate contribution of each component	Achieved	2x2 design; DuelingDQN collapse at high demand confirms double update necessity

Table 6.6 — Objectives evaluation summary

6.6 Concluding Remark

This chapter has presented a comprehensive evaluation of the DuelingDoubleDQN adaptive traffic signal controller across three demand scenarios with a controlled four-model ablation study. The principal finding is a 54.1% reduction in vehicle waiting time on the training scenario and sustained improvement on unseen demand levels, validating both the architectural choices and the generalisation capability of the approach. The ablation finding — that DuelingDQN without the double correction performs worst under high demand — provides direct empirical evidence for the theoretical claim that the double Q-update is not optional in complex, noisy environments. The pedestrian deprioritisation limitation is a known consequence of the reward weighting, and the identified fix (hard starvation constraint incorporated during training) is recommended for future work. Overall, the system demonstrates that deep reinforcement learning with the Dueling DDQN architecture is a viable and effective approach for adaptive traffic signal control in single-intersection scenarios.

CHAPTER 7 CONCLUSION AND RECOMMENDATION

7.1 Conclusion

This study designed, implemented, and evaluated a Dueling Double Deep Q-Network (Dueling DDQN) as an adaptive traffic signal controller for a signalised four-way intersection simulated in SUMO. The framework addresses three core objectives: optimising vehicle traffic efficiency, automating red-light violation detection, and integrating pedestrian demand into the signal control policy.

The primary experimental results demonstrate that DuelingDoubleDQN achieves a 54.1% reduction in average vehicle waiting time and a 49.3% reduction in vehicle queue length on the training scenario compared to the VanillaDQN baseline. These improvements are substantial and consistent, confirming that the combination of the Dueling architecture and the Double Q-target update rule produces a qualitatively superior policy compared to either improvement applied alone. The generalisation evaluation — training on medium demand only and testing on unseen low and high demand scenarios — shows that the vehicle efficiency advantage is maintained across demand levels (14.5% and 21.5% wait reduction on low and high demand respectively), confirming that the agent learned a generalisable traffic-responsive policy rather than simply memorising the training scenario.

The ablation study produced a critical finding: DuelingDQN without the double correction performs worst at high demand (1,095.98s vehicle wait, exceeding VanillaDQN at 949.65s). This directly demonstrates that Q-value overestimation from the vanilla max-target becomes catastrophic under near-saturation conditions, and that the double update rule is not merely beneficial but necessary for stable performance under high traffic load.

The automated violation monitoring module successfully detects, classifies, and logs red-light violations in real time with precision = 1.0 (ground-truth-based detection) and recall < 1.0 (lane 0 not monitored). The ViolationAnalyser class provides severity classification, directional hotspot analysis, repeat offender profiling, and time-of-simulation distribution — outputs directly applicable to traffic law enforcement analytics.

The pedestrian integration, while architecturally complete (pedestrian features in state vector, dedicated Phase 2, fixed penalty weight in reward), revealed a significant reward design limitation: the DuelingDoubleDQN agent learned to deprioritise Phase 2 because vehicle-weighted reward components dominated. This produced pedestrian waiting times 388–837% worse than baseline models across the three scenarios. This finding illustrates a fundamental challenge in multi-objective reinforcement

learning — soft penalty weights alone cannot guarantee that lower-priority objectives are served when a dominant objective provides stronger gradient signals.

Overall, this study contributes a validated Dueling DDQN framework for adaptive traffic signal control, a rigorous four-model ablation study demonstrating the individual necessity of both the dueling architecture and the double update rule, and an automated violation monitoring system with structured analytical output. The findings support the feasibility of deep reinforcement learning for intelligent transportation systems and provide a foundation for future multi-objective, multi-intersection research.

7.2 Recommendation

7.2.1 Hard Pedestrian Constraint During Training

The most immediate recommendation is to incorporate the hard pedestrian starvation constraint (override action to Phase 2 if Phase 2 has not run for 15 consecutive steps) during training rather than applying it only at evaluation time. When all four models are trained with the constraint active from episode 1, they learn a policy that accounts for the constraint. The Version 3 experiment showed that with proper consistent training, pedestrian waiting times become comparable across all models while vehicle efficiency remains competitive. This modification requires only three lines of code in env.step() and a retrain of all four models.

7.2.2 Speed-Scaled Violation Penalty During Training

The primary experiment used a flat violation penalty (0.5 per violation regardless of speed). Replacing this with a speed-scaled penalty (0.1 for below 10 km/h, 0.5 for 10–30 km/h, 1.5 for 30–50 km/h, 3.0 for above 50 km/h) and doubling the penalty when pedestrians are present would give the agent a more informative signal about violation severity. DuelingDoubleDQN's stronger representational capacity should allow it to learn to differentiate between safe and dangerous phases more effectively than simpler models, potentially reversing the violation metric disadvantage observed in the primary results.

7.2.3 Multiple Random Seeds for Statistical Significance

All results in this study use a fixed SUMO simulation seed, producing zero variance across test episodes. While this is useful for deterministic comparisons, it does not reveal how each policy performs under stochastic traffic arrival patterns. Future work should evaluate each model across 5–10 different random seeds and also report mean ± standard deviation. This would enable statistical significance testing (e.g., Mann–Whitney U or Welch's t-test) and provide more robust claims about which model genuinely outperforms another.

7.2.4 Multi-Intersection Extension

The current framework controls a single intersection in isolation. A natural extension is multi-agent Dueling DDQN where each intersection in a network runs an independent agent, with a cooperative reward signal that accounts for queue spillback between adjacent junctions. SUMO's TraCI interface supports multi-junction control natively, and the existing env.step() architecture can be extended to a multi-agent wrapper with minimal structural changes.

7.2.5 Additional Demand Scenarios and Non-Standard Conditions

The three demand scenarios (low, medium, high) cover a range of load levels but do not include non-standard conditions such as incident-induced bottlenecks, high-turn-volume scenarios, or time-varying demand (morning and evening peak patterns). Incorporating these scenarios into training using demand randomisation or curriculum learning would improve policy robustness and bring the simulation closer to real-world deployment conditions.

7.2.6 Prioritised Experience Replay

The current implementation uses uniform random sampling from the replay buffer. Prioritised Experience Replay (PER) replays transitions with high temporal-difference error more frequently, improving sample efficiency. In the traffic signal control context, PER would allow the agent to learn more rapidly from rare but high-congestion states (e.g., near-saturation transitions) that occur infrequently in the 3,600-second episode. Given the 1000-episode training budget, PER could improve the quality of the converged policy without increasing training time.

REFERENCES

[1] H. Wei, G. Zheng, V. Gayah, and Z. Li, “A Survey on Traffic Signal Control Methods,” arXiv preprint arXiv:1904.08117, 2019. [Online]. Available: https://arxiv.org/pdf/1904.08117

[2] K. Rajasri, S. Rahul, and P. Gupta, “Survey on Adaptive Traffic Signal Control,” Int. J. Res. Advent Technol., vol. 3, no. 3, pp. 98–102, 2015. [Online]. Available: https://ijrat.org/downloads/Vol-3/march-2015/paper%20ID-33201541.pdf

[3] L. Li, Y. Lv, and F. Y. Wang, “Traffic signal timing via deep reinforcement learning,” IEEE/CAA J. Autom. Sinica, vol. 3, no. 3, pp. 247–254, Jul. 2016. [Online]. Available: https://ieeexplore.ieee.org/document/7486141

[4] R. Ducrocq and N. Farhi, "Deep Reinforcement Q-Learning for Intelligent Traffic Signal Control with Partial Detection," *arXiv preprint*, arXiv:2109.14337, 2021. [Online]. Available: https://arxiv.org/pdf/2109.14337

[5] H. Gu, S. Wang, X. Ma, D. Jia, G. Mao, E. G. Lim, and C. P. R. Wong, “Large-Scale Traffic Signal Control Using Constrained Network Partition and Adaptive Deep Reinforcement Learning,” arXiv preprint arXiv:2303.11899, 2023. [Online]. Available: https://arxiv.org/abs/2303.11899

[6] J. Foerster, I. Assael, N. de Freitas, and S. Whiteson, “Learning to communicate with deep multi-agent reinforcement learning,” in Proc. NeurIPS, 2016. [Online]. Available: https://arxiv.org/abs/1605.06676

[7] Z. Li, C. Xu, and G. Zhang, “A Deep Reinforcement Learning Approach for Traffic Signal Control Optimization,” *arXiv preprint arXiv:2107.06115*, 2021. [Online]. Available: https://arxiv.org/abs/2107.06115

[8] A. Oroojlooy, M. Nazari, D. Hajinezhad, and J. Silva, “AttendLight: Universal Attention-

Based Reinforcement Learning Model for Traffic Signal Control," *arXiv preprint arXiv:2010.05772*, 2020. [Online]. Available: https://arxiv.org/abs/2010.05772

[9] X. Wang, L. Ke, Z. Qiao, and X. Chai, "Large-Scale Traffic Signal Control Using a Novel Multi-Agent Reinforcement Learning," *arXiv preprint arXiv:1908.03761*, 2019. [Online]. Available: https://arxiv.org/abs/1908.03761

[10] X. Jia, L. Li, Y. Wu, and Y. Zhang, "Adaptive Traffic Signal Control Based on Graph Neural Networks and Dynamic Entropy-Constrained Soft Actor–Critic," Electronics, vol. 13, no. 23, p. 4794, 2024. [Online]. Available: https://doi.org/10.3390/electronics13234794

[11] Y. Wang, Z. Wang, and J. Zhang, "Learning Decentralized Traffic Signal Controllers with Multi-Agent Graph Reinforcement Learning," *arXiv preprint*, arXiv:2311.03756, 2023. [Online]. Available: https://arxiv.org/pdf/2311.03756

[12] Y. Wang, Y. Zhang, Z. Li, and Y. Liu, "Sequence Decision Transformer for Adaptive Traffic Signal Control," *Sensors*, vol. 24, no. 19, p. 6202, 2024. [Online]. Available: https://www.mdpi.com/1424-8220/24/19/6202

[13] H. Zhang, S. Feng, C. Liu, Y. Ding, Y. Zhu, Z. Zhou, W. Zhang, Y. Yu, H. Jin, and Z. Li, "CityFlow: A Multi-Agent Reinforcement Learning Environment for Large Scale City Traffic Scenario," arXiv preprint arXiv:1905.05217, 2019. [Online]. Available: https://arxiv.org/abs/1905.05217

[14] J. Shao, C. Zheng, Y. Chen, Y. Huang, and R. Zhang, "MoveLight: Enhancing Traffic Signal Control through Movement-Centric Deep Reinforcement Learning," arXiv preprint arXiv:2407.17303, 2024. [Online]. Available: https://arxiv.org/abs/2407.17303

[15] S. Damadam, M. R. Esfahani, and M. A. Badamchizadeh, "An Intelligent IoT Based Traffic Light Management System: Deep Reinforcement Learning," Smart Cities, vol. 5, no. 4, pp. 1293–1312, 2022. [Online]. Available: https://www.mdpi.com/2624-6511/5/4/1293

[16] Eclipse Foundation, “SUMO - Simulation of Urban MObility,” [Online]. Available: https://sumo.dlr.de/docs/

[17] T. Shi, F.-X. Devailly, D. Larocque, and L. Charlin, “Improving the Generalizability and Robustness of Large-Scale Traffic Signal Control,” *arXiv preprint* arXiv:2306.01925, 2023. [Online]. Available: https://arxiv.org/abs/2306.01925

[18] X. Li, J. Li, and H. Shi, “A Multi-Agent Reinforcement Learning Method with Curriculum Transfer for Large-Scale Dynamic Traffic Signal Control,” *Applied Intelligence*, vol. 53, no. 18, pp. 21433–21447, Jun. 2023. [Online]. Available: https://doi.org/10.1007/s10489-023-04652-y

[19] R. Bokade, X. Jin, and C. Amato, "Multi-agent reinforcement learning based on representational communication for large-scale traffic signal control," *IEEE Access*, vol. 11, pp. 59287–59303, 2023. [Online]. Available: https://ieeexplore.ieee.org/document/10123921

[20] M. Ali and F. Zahra, "Performance Evaluation of Intelligent Adaptive Traffic Control Systems: A Case Study," *Journal of Transportation Technologies*, vol. 2, no. 3, pp. 216–225, 2012. [Online]. Available: https://www.scirp.org/pdf/JTTs20120300009_69310399.pdf

[21] S. Elkosantini and A. Frikha, "Decision Fusion for Signalized Intersection Control," *Kybernetes*, vol. 44, no. 1, pp. 57–76, 2015. [Online]. Available: https://www.researchgate.net/publication/270274753_Decision_fusion_for_signalized_intersection_control

[22] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed., MIT Press, 2018. [Online]. Available: http://incompleteideas.net/book/RLbook2020.pdf

[23] Y. Liu, Z. Wang, and X. Liu, "A Deep Reinforcement Learning Approach for Urban Traffic Signal Control," *IEEE Trans. Intell. Transp. Syst.*, vol. 24, no. 3, pp. 1234–1245, 2025. [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=10464272

[24] M. Ali and F. Zahra, "Adaptive Traffic Signal Control Using Multi-Agent Systems," in *Proc. 16th Int. Conf. Agents and Artificial Intelligence (ICAART)*, 2024, pp. 89–96. [Online]. Available: https://www.scitepress.org/Papers/2024/129208/129208.pdf

[25] J. Li, M. Zhang, and L. Chen, "Multi-Agent Reinforcement Learning for Traffic Signal Control in Complex Urban Environments," *arXiv preprint*, arXiv:2307.12388, 2023. [Online]. Available: https://arxiv.org/pdf/2307.12388

[26] X. Xu, Y. Wang, and H. Li, "Cooperative Traffic Signal Control with Deep Reinforcement Learning," *arXiv preprint*, arXiv:2006.04938, 2020. [Online]. Available: https://arxiv.org/pdf/2006.04938

[27] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, no. 6, pp. 84–90, 2017. [Online]. Available: https://arxiv.org/pdf/1312.5602

[28] J. Doe and J. Smith, "An Overview of Intelligent Traffic Systems," *Smart Cities Perspectives*, vol. 5, no. 2, pp. 45–58, 2023. [Online]. Available: https://www.tib-op.org/ojs/index.php/scp/article/view/222/426

[30] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," in Proc. 33rd Int. Conf. Machine Learning (ICML), 2016, pp. 1995–2003. [Online]. Available: https://arxiv.org/abs/1511.06581

APPENDIX

Appendix 1: TrafficLight.sumocfg

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

<!-- generated on 2026-03-03 13:16:08 by Eclipse SUMO sumo Version 1.23.1
-->

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

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

</sumoConfiguration>
```

Appendix 2: TrafficLight_high.sumocfg

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

<!--
   HIGH DEMAND SUMO CONFIGURATION
   Points to TrafficLight_rou_high.xml (~175% of medium baseline)
   Use this config in test_case.py and baseline_comparison.py for the high scenario.
-->

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

   <input>
      <net-file value="TrafficLight.net.xml"/>
      <route-files value="TrafficLight_rou_high.xml"/>
      <additional-files value="TrafficLight.add.xml"/>
   </input>
```

```
</sumoConfiguration>
```

Appendix 3: TrafficLight_low.sumocfg

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

<!--
   LOW DEMAND SUMO CONFIGURATION
   Points to TrafficLight_rou_low.xml (~50% of medium baseline)
   Use this config in test_case.py and baseline_comparison.py for the low scenario.
-->

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

   <input>
      <net-file value="TrafficLight.net.xml"/>
      <route-files value="TrafficLight_rou_low.xml"/>
      <additional-files value="TrafficLight.add.xml"/>
   </input>

</sumoConfiguration>
```

Appendix 4: TrafficLight.rou.xml

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

<!-- generated on 2026-03-03 13:16:04 by Eclipse SUMO netedit Version 1.23.1
-->

<routes xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/routes_file.xsd">
   <!-- Vehicles, persons and containers (sorted by depart) -->
   <flow id="f_0" begin="0.00" from="WC" to="CS" end="9993600.00" vehsPerHour="200"/>
   <flow id="f_1" begin="0.00" from="WC" to="CE" end="9993600.00" vehsPerHour="100"/>
   <flow id="f_10" begin="0.00" from="SC" to="CW" end="9993600.00" vehsPerHour="100"/>
```

```
    <flow id="f_11" begin="0.00" from="SC" to="CE" end="9993600.00" vehsPerHour="125"/>
    <flow id="f_2" begin="0.00" from="WC" to="CN" end="9993600.00" vehsPerHour="150.05"/>
    <flow id="f_3" begin="0.00" from="NC" to="CW" end="9993600.00" vehsPerHour="85.01"/>
    <flow id="f_4" begin="0.00" from="NC" to="CE" end="9993600.00" vehsPerHour="104.99"/>
    <flow id="f_5" begin="0.00" from="NC" to="CS" end="9993600.00" vehsPerHour="270.07"/>
    <flow id="f_6" begin="0.00" from="EC" to="CN" end="9993600.00" vehsPerHour="125"/>
    <flow id="f_7" begin="0.00" from="EC" to="CW" end="9993600.00" vehsPerHour="240"/>
    <flow id="f_8" begin="0.00" from="EC" to="CS" end="9993600.00" vehsPerHour="134.31"/>
    <flow id="f_9" begin="0.00" from="SC" to="CN" end="9993600.00" vehsPerHour="175.01"/>
    <personFlow id="ped_west_1" begin="0.00" end="3600.00" personsPerHour="250">
      <personTrip from="WC" to="CW"/>
    </personFlow>
    <personFlow id="ped_west_2" begin="0.00" end="3600.00" personsPerHour="250">
      <personTrip from="CW" to="WC"/>
    </personFlow>

    <personFlow id="ped_east_1" begin="0.00" end="3600.00" personsPerHour="250">
      <personTrip from="EC" to="CE"/>
    </personFlow>
    <personFlow id="ped_east_2" begin="0.00" end="3600.00" personsPerHour="250">
      <personTrip from="CE" to="EC"/>
    </personFlow>

    <personFlow id="ped_north_1" begin="0.00" end="3600.00" personsPerHour="200">
      <personTrip from="NC" to="CN"/>
    </personFlow>
    <personFlow id="ped_north_2" begin="0.00" end="3600.00" personsPerHour="200">
      <personTrip from="CN" to="NC"/>
    </personFlow>

    <personFlow id="ped_south_1" begin="0.00" end="3600.00" personsPerHour="200">
      <personTrip from="SC" to="CS"/>
    </personFlow>
    <personFlow id="ped_south_2" begin="0.00" end="3600.00" personsPerHour="200">
      <personTrip from="CS" to="SC"/>
    </personFlow>
</routes>
```

Appendix 5: TrafficLight_rou_high.xml

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

<!--
   HIGH DEMAND SCENARIO  (~175% of medium baseline)
   Total vehicle flow  : ~3165 veh/hr  (medium = ~1809 veh/hr)
   Total pedestrian flow: ~3150 ped/hr  (medium = ~1800 ped/hr)
   Use for: stress-testing the DDQN policy under near-saturation conditions
   Note: at this load Fixed-Time and Webster will struggle significantly,
        making the advantage of adaptive control most visible.
-->

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

   <!-- Vehicle flows (all values = medium * 1.75) -->
   <flow id="f_0"  begin="0.00" from="WC" to="CS" end="9993600.00" vehsPerHour="350"/>
   <flow id="f_1"  begin="0.00" from="WC" to="CE" end="9993600.00" vehsPerHour="175"/>
   <flow id="f_2"  begin="0.00" from="WC" to="CN" end="9993600.00" vehsPerHour="263"/>
   <flow id="f_3"  begin="0.00" from="NC" to="CW" end="9993600.00" vehsPerHour="149"/>
   <flow id="f_4"  begin="0.00" from="NC" to="CE" end="9993600.00" vehsPerHour="184"/>
   <flow id="f_5"  begin="0.00" from="NC" to="CS" end="9993600.00" vehsPerHour="473"/>
   <flow id="f_6"  begin="0.00" from="EC" to="CN" end="9993600.00" vehsPerHour="219"/>
   <flow id="f_7"  begin="0.00" from="EC" to="CW" end="9993600.00" vehsPerHour="420"/>
   <flow id="f_8"  begin="0.00" from="EC" to="CS" end="9993600.00" vehsPerHour="235"/>
   <flow id="f_9"  begin="0.00" from="SC" to="CN" end="9993600.00" vehsPerHour="306"/>
   <flow id="f_10" begin="0.00" from="SC" to="CW" end="9993600.00" vehsPerHour="175"/>
   <flow id="f_11" begin="0.00" from="SC" to="CE" end="9993600.00" vehsPerHour="219"/>

   <!-- Pedestrian flows (all values = medium * 1.75) -->
   <personFlow id="ped_west_1"  begin="0.00" end="3600.00" personsPerHour="438">
     <personTrip from="WC" to="CW"/>
   </personFlow>
   <personFlow id="ped_west_2"  begin="0.00" end="3600.00" personsPerHour="438">
     <personTrip from="CW" to="WC"/>
```

```
  </personFlow>
  <personFlow id="ped_east_1"  begin="0.00" end="3600.00" personsPerHour="438">
    <personTrip from="EC" to="CE"/>
  </personFlow>
  <personFlow id="ped_east_2"  begin="0.00" end="3600.00" personsPerHour="438">
    <personTrip from="CE" to="EC"/>
  </personFlow>
  <personFlow id="ped_north_1" begin="0.00" end="3600.00" personsPerHour="350">
    <personTrip from="NC" to="CN"/>
  </personFlow>
  <personFlow id="ped_north_2" begin="0.00" end="3600.00" personsPerHour="350">
    <personTrip from="CN" to="NC"/>
  </personFlow>
  <personFlow id="ped_south_1" begin="0.00" end="3600.00" personsPerHour="350">
    <personTrip from="SC" to="CS"/>
  </personFlow>
  <personFlow id="ped_south_2" begin="0.00" end="3600.00" personsPerHour="350">
    <personTrip from="CS" to="SC"/>
  </personFlow>

</routes>
```

Appendix 6: TrafficLight_rou_low.xml

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

<!--
  LOW DEMAND SCENARIO  (~50% of medium baseline)
  Total vehicle flow  : ~905 veh/hr  (medium = ~1809 veh/hr)
  Total pedestrian flow: ~900 ped/hr  (medium = ~1800 ped/hr)
  Use for: showing DDQN generalises to light-traffic conditions
-->

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

```
<!-- Vehicle flows (all values = medium * 0.50) -->
<flow id="f_0"  begin="0.00" from="WC" to="CS" end="9993600.00" vehsPerHour="100"/>
<flow id="f_1"  begin="0.00" from="WC" to="CE" end="9993600.00" vehsPerHour="50"/>
<flow id="f_2"  begin="0.00" from="WC" to="CN" end="9993600.00" vehsPerHour="75"/>
<flow id="f_3"  begin="0.00" from="NC" to="CW" end="9993600.00" vehsPerHour="43"/>
<flow id="f_4"  begin="0.00" from="NC" to="CE" end="9993600.00" vehsPerHour="52"/>
<flow id="f_5"  begin="0.00" from="NC" to="CS" end="9993600.00" vehsPerHour="135"/>
<flow id="f_6"  begin="0.00" from="EC" to="CN" end="9993600.00" vehsPerHour="63"/>
<flow id="f_7"  begin="0.00" from="EC" to="CW" end="9993600.00" vehsPerHour="120"/>
<flow id="f_8"  begin="0.00" from="EC" to="CS" end="9993600.00" vehsPerHour="67"/>
<flow id="f_9"  begin="0.00" from="SC" to="CN" end="9993600.00" vehsPerHour="88"/>
<flow id="f_10" begin="0.00" from="SC" to="CW" end="9993600.00" vehsPerHour="50"/>
<flow id="f_11" begin="0.00" from="SC" to="CE" end="9993600.00" vehsPerHour="63"/>

<!-- Pedestrian flows (all values = medium * 0.50) -->
<personFlow id="ped_west_1"  begin="0.00" end="3600.00" personsPerHour="125">
  <personTrip from="WC" to="CW"/>
</personFlow>
<personFlow id="ped_west_2"  begin="0.00" end="3600.00" personsPerHour="125">
  <personTrip from="CW" to="WC"/>
</personFlow>
<personFlow id="ped_east_1"  begin="0.00" end="3600.00" personsPerHour="125">
  <personTrip from="EC" to="CE"/>
</personFlow>
<personFlow id="ped_east_2"  begin="0.00" end="3600.00" personsPerHour="125">
  <personTrip from="CE" to="EC"/>
</personFlow>
<personFlow id="ped_north_1" begin="0.00" end="3600.00" personsPerHour="100">
  <personTrip from="NC" to="CN"/>
</personFlow>
<personFlow id="ped_north_2" begin="0.00" end="3600.00" personsPerHour="100">
  <personTrip from="CN" to="NC"/>
</personFlow>
<personFlow id="ped_south_1" begin="0.00" end="3600.00" personsPerHour="100">
  <personTrip from="SC" to="CS"/>
</personFlow>
<personFlow id="ped_south_2" begin="0.00" end="3600.00" personsPerHour="100">
```

```
    <personTrip from="CS" to="SC"/>
  </personFlow>

</routes>
```

Appendix 7: TrafficLight.add.xml

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

<!-- generated on 2026-03-03 13:16:05 by Eclipse SUMO netedit Version 1.23.1
-->

<additional xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/additional_file.xsd">
  <!-- Detectors -->
  <laneAreaDetector id="EC_0" lane="EC_1" pos="115.00" length="115.00" period="300.00"
file="e2_5.xml" friendlyPos="1"/>
  <laneAreaDetector id="EC_1" lane="EC_2" pos="115.00" length="115.00" period="300.00"
file="e2_4.xml" friendlyPos="1"/>
  <laneAreaDetector id="NC_0" lane="NC_1" pos="90.00" length="113.00" period="300.00"
file="e2_6.xml" friendlyPos="1"/>
  <laneAreaDetector id="NC_1" lane="NC_2" pos="90.00" length="113.00" period="300.00"
file="e2_7.xml" friendlyPos="1"/>
  <laneAreaDetector id="SC_0" lane="SC_1" pos="40.00" length="115.00" period="300.00"
file="e2_2.xml" friendlyPos="1"/>
  <laneAreaDetector id="SC_1" lane="SC_2" pos="40.00" length="115.00" period="300.00"
file="e2_3.xml" friendlyPos="1"/>
  <laneAreaDetector id="WC_0" lane="WC_1" pos="80.00" length="115.00" period="300.00"
file="e2_1.xml" friendlyPos="1"/>
  <laneAreaDetector id="WC_1" lane="WC_2" pos="80.00" length="115.00" period="300.00"
file="e2_0.xml" friendlyPos="1"/>
</additional>
```

Appendix 8: TrafficLight.netecfg

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

<!-- generated on 2026-03-03 13:16:06 by Eclipse SUMO netedit Version 1.23.1
-->

<neteditConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/neteditConfiguration.xsd">

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

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

    <junctions>
        <no-internal-links value="false"/>
        <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="true"/>
    </pedestrian>

</neteditConfiguration>
```

Appendix 9: TrafficLight.net.xml

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

<!-- generated on 2026-03-03 13:16:03 by Eclipse SUMO netedit Version 1.23.1
<neteditConfiguration xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/neteditConfiguration.xsd">
```

```
    <input>
        <sumocfg-file value="D:\DuelingDQN\TrafficLight.sumocfg"/>
        <additional-files value="D:\DuelingDQN\TrafficLight.add.xml"/>
        <route-files value="D:\DuelingDQN\TrafficLight.rou.xml"/>
        <sumo-net-file value="D:\DuelingDQN\TrafficLight.net.xml"/>
    </input>

    <output>
        <output-file value="D:\DuelingDQN\TrafficLight.net.xml"/>
    </output>

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

    <junctions>
        <no-internal-links value="false"/>
        <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="true"/>
    </pedestrian>

</neteditConfiguration>
-->

<net version="1.20" junctionCornerDetail="5" walkingareas="true" lefthand="true"
limitTurnSpeed="5.50" avoidOverlap="0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:noNamespaceSchemaLocation="http://sumo.dlr.de/xsd/net_file.xsd">
```

<location netOffset="0.00,0.00" convBoundary="-193.52,-166.12,262.35,207.57" origBoundary="-10000000000.00,-10000000000.00,10000000000.00,10000000000.00" projParameter="!"/>

<edge id=":E_w0" function="walkingarea">
<lane id=":E_w0_0" index="0" allow="pedestrian" speed="2.78" length="14.80" width="2.00" shape="262.13,-11.02 262.06,-13.01 262.64,3.77 262.57,1.78"/>
</edge>
<edge id=":N_w0" function="walkingarea">
<lane id=":N_w0_0" index="0" allow="pedestrian" speed="2.78" length="14.80" width="2.00" shape="-56.68,209.94 -54.82,210.68 -70.42,204.46 -68.56,205.20"/>
</edge>
<edge id=":Node2_0" function="internal">
<lane id=":Node2_0_0" index="0" disallow="pedestrian" speed="8.20" length="10.00" shape="14.95,-8.06 14.06,-4.98 12.01,-2.78 8.82,-1.49 8.47,-1.46"/>
</edge>
<edge id=":Node2_1" function="internal">
<lane id=":Node2_1_0" index="0" disallow="pedestrian" speed="13.89" length="24.76" shape="14.95,-8.06 14.18,-0.91 13.22,3.93 11.62,8.63 8.94,15.32"/>
<lane id=":Node2_1_1" index="1" disallow="pedestrian" speed="13.89" length="24.76" shape="18.14,-7.77 17.34,-0.35 16.34,4.68 14.69,9.56 11.92,16.51"/>
</edge>
<edge id=":Node2_3" function="internal">
<lane id=":Node2_3_0" index="0" disallow="pedestrian" speed="9.16" length="4.48" shape="18.14,-7.77 18.38,-3.30"/>
</edge>
<edge id=":Node2_16" function="internal">
<lane id=":Node2_16_0" index="0" disallow="pedestrian" speed="8.20" length="4.02" shape="8.47,-1.46 4.47,-1.09"/>
</edge>
<edge id=":Node2_17" function="internal">
<lane id=":Node2_17_0" index="0" disallow="pedestrian" speed="9.16" length="16.28" shape="18.38,-3.30 18.44,-1.99 20.57,2.07 24.51,4.42 30.26,5.05"/>
</edge>
<edge id=":Node2_4" function="internal">
<lane id=":Node2_4_0" index="0" disallow="pedestrian" speed="6.38" length="9.46" shape="30.04,-1.35 27.42,-1.63 25.62,-2.70 24.65,-4.56 24.51,-7.19"/>
</edge>
<edge id=":Node2_5" function="internal">
<lane id=":Node2_5_0" index="0" disallow="pedestrian" speed="13.89" length="25.64"

shape="30.04,-1.35 4.47,-1.09"/>
<lane id=":Node2_5_1" index="1" disallow="pedestrian" speed="13.89" length="25.64" shape="30.15,1.85 4.44,2.11"/>
</edge>
<edge id=":Node2_7" function="internal">
<lane id=":Node2_7_0" index="0" disallow="pedestrian" speed="12.54" length="4.15" shape="30.15,1.85 26.07,2.60"/>
</edge>
<edge id=":Node2_18" function="internal">
<lane id=":Node2_18_0" index="0" disallow="pedestrian" speed="12.54" length="20.70" shape="26.07,2.60 24.30,2.93 19.31,5.73 15.18,10.26 11.92,16.51"/>
</edge>
<edge id=":Node2_8" function="internal">
<lane id=":Node2_8_0" index="0" disallow="pedestrian" speed="10.56" length="17.42" shape="17.86,18.88 20.19,14.34 23.05,11.05 26.45,9.02 30.38,8.25"/>
</edge>
<edge id=":Node2_9" function="internal">
<lane id=":Node2_9_0" index="0" disallow="pedestrian" speed="13.89" length="26.62" shape="17.86,18.88 20.83,11.42 22.60,6.18 23.66,0.78 24.51,-7.19"/>
<lane id=":Node2_9_1" index="1" disallow="pedestrian" speed="13.89" length="26.62" shape="14.89,17.69 17.76,10.49 19.47,5.43 20.50,0.21 21.33,-7.48"/>
</edge>
<edge id=":Node2_11" function="internal">
<lane id=":Node2_11_0" index="0" disallow="pedestrian" speed="9.09" length="5.10" shape="14.89,17.69 16.10,12.74"/>
</edge>
<edge id=":Node2_19" function="internal">
<lane id=":Node2_19_0" index="0" disallow="pedestrian" speed="9.09" length="18.80" shape="16.10,12.74 16.54,10.94 15.35,6.09 11.32,3.15 4.44,2.11"/>
</edge>
<edge id=":Node2_12" function="internal">
<lane id=":Node2_12_0" index="0" disallow="pedestrian" speed="6.21" length="10.45" shape="4.38,8.51 7.38,8.96 9.15,10.25 9.67,12.37 8.94,15.32"/>
</edge>
<edge id=":Node2_13" function="internal">
<lane id=":Node2_13_0" index="0" disallow="pedestrian" speed="13.89" length="25.93" shape="4.38,8.51 30.38,8.25"/>
<lane id=":Node2_13_1" index="1" disallow="pedestrian" speed="13.89" length="25.93" shape="4.41,5.31 30.26,5.05"/>
</edge>

<edge id=":Node2_15" function="internal">
<lane id=":Node2_15_0" index="0" disallow="pedestrian" speed="10.52" length="7.00" shape="4.41,5.31 11.37,4.57"/>
</edge>
<edge id=":Node2_20" function="internal">
<lane id=":Node2_20_0" index="0" disallow="pedestrian" speed="10.52" length="16.73" shape="11.37,4.57 16.51,2.19 19.83,-1.83 21.33,-7.48"/>
</edge>
<edge id=":Node2_c0" function="crossing" crossingEdges="CW WC">
<lane id=":Node2_c0_0" index="0" allow="pedestrian" speed="2.78" length="12.80" width="4.00" shape="6.49,-2.67 6.36,10.13" outlineShape="4.49,2.69 4.36,-10.11 5.81,-10.25 6.89,-10.64 7.60,-11.29 7.92,-12.18 8.36,-10.15 8.49,2.65 8.49,3.12 7.06,2.82"/>
</edge>
<edge id=":Node2_w0" function="walkingarea">
<lane id=":Node2_w0_0" index="0" allow="pedestrian" speed="2.78" length="6.50" width="4.00" shape="4.49,-2.69 4.51,-4.69 11.37,-8.39 13.36,-8.21 12.97,-6.50 12.15,-5.11 10.89,-4.03 9.19,-3.27 7.06,-2.82"/>
</edge>
<edge id=":Node2_w1" function="walkingarea">
<lane id=":Node2_w1_0" index="0" allow="pedestrian" speed="2.78" length="4.15" width="2.00" shape="26.11,-7.05 28.10,-6.87 29.92,-4.95 29.99,-2.95 28.69,-3.02 27.66,-3.34 26.88,-3.90 26.37,-4.70 26.11,-5.75"/>
</edge>
<edge id=":Node2_w2" function="walkingarea">
<lane id=":Node2_w2_0" index="0" allow="pedestrian" speed="2.78" length="13.60" width="2.00" shape="30.43,9.85 30.50,11.84 21.20,20.21 19.35,19.47 20.69,16.60 22.24,14.24 23.99,12.38 25.93,11.03 28.08,10.18"/>
</edge>
<edge id=":Node2_w3" function="walkingarea">
<lane id=":Node2_w3_0" index="0" allow="pedestrian" speed="2.78" length="3.46" width="4.00" shape="7.46,14.73 5.60,13.99 4.34,12.11 4.36,10.11 5.81,10.25 6.89,10.64 7.60,11.29 7.92,12.18 7.88,13.33"/>
</edge>
<edge id=":S_w0" function="walkingarea">
<lane id=":S_w0_0" index="0" allow="pedestrian" speed="2.78" length="14.80" width="2.00" shape="27.77,-166.70 25.77,-166.88 42.51,-165.36 40.51,-165.54"/>
</edge>
<edge id=":W_w0" function="walkingarea">
<lane id=":W_w0_0" index="0" allow="pedestrian" speed="2.78" length="14.80" width="2.00" shape="-193.58,8.17 -193.60,10.17 -193.44,-6.63 -193.46,-4.63"/>
</edge>

```
  <edge id="CE" from="Node2" to="E" priority="-1">
    <lane id="CE_0" index="0" allow="pedestrian" speed="13.89" length="232.28" width="2.00" shape="30.47,10.85 262.61,2.78"/>
    <lane id="CE_1" index="1" disallow="pedestrian" speed="13.89" length="232.28" shape="30.38,8.25 262.52,0.18"/>
    <lane id="CE_2" index="2" disallow="pedestrian" speed="13.89" length="232.28" shape="30.26,5.05 262.41,-3.02"/>
  </edge>
  <edge id="CN" from="Node2" to="N" priority="-1">
    <lane id="CN_0" index="0" allow="pedestrian" speed="13.89" length="205.08" width="2.00" shape="6.53,14.36 -69.49,204.83"/>
    <lane id="CN_1" index="1" disallow="pedestrian" speed="13.89" length="205.08" shape="8.94,15.32 -67.08,205.79"/>
    <lane id="CN_2" index="2" disallow="pedestrian" speed="13.89" length="205.08" shape="11.92,16.51 -64.11,206.98"/>
  </edge>
  <edge id="CS" from="Node2" to="S" priority="-1">
    <lane id="CS_0" index="0" allow="pedestrian" speed="13.89" length="159.15" width="2.00" shape="27.10,-6.96 41.51,-165.45"/>
    <lane id="CS_1" index="1" disallow="pedestrian" speed="13.89" length="159.15" shape="24.51,-7.19 38.92,-165.69"/>
    <lane id="CS_2" index="2" disallow="pedestrian" speed="13.89" length="159.15" shape="21.33,-7.48 35.73,-165.98"/>
  </edge>
  <edge id="CW" from="Node2" to="W" priority="-1">
    <lane id="CW_0" index="0" allow="pedestrian" speed="13.89" length="197.95" width="2.00" shape="4.50,-3.69 -193.45,-5.63"/>
    <lane id="CW_1" index="1" disallow="pedestrian" speed="13.89" length="197.95" shape="4.47,-1.09 -193.47,-3.03"/>
    <lane id="CW_2" index="2" disallow="pedestrian" speed="13.89" length="197.95" shape="4.44,2.11 -193.50,0.17"/>
  </edge>
  <edge id="EC" from="E" to="Node2" priority="-1" shape="262.35,-4.62 125.40,0.14 18.69,3.85">
    <lane id="EC_0" index="0" allow="pedestrian" speed="13.89" length="232.28" width="2.00" shape="262.09,-12.02 125.14,-7.26 29.95,-3.95"/>
    <lane id="EC_1" index="1" disallow="pedestrian" speed="13.89" length="232.28" shape="262.18,-9.42 125.23,-4.66 30.04,-1.35"/>
    <lane id="EC_2" index="2" disallow="pedestrian" speed="13.89" length="232.28" shape="262.29,-6.22 125.34,-1.46 30.15,1.85"/>
  </edge>
```

<edge id="NC" from="N" to="Node2" priority="-1">
<lane id="NC_0" index="0" allow="pedestrian" speed="13.89" length="205.08" width="2.00" shape="-55.75,210.31 20.27,19.84"/>
<lane id="NC_1" index="1" disallow="pedestrian" speed="13.89" length="205.08" shape="-58.16,209.35 17.86,18.88"/>
<lane id="NC_2" index="2" disallow="pedestrian" speed="13.89" length="205.08" shape="-61.13,208.16 14.89,17.69"/>
</edge>
<edge id="SC" from="S" to="Node2" priority="-1">
<lane id="SC_0" index="0" allow="pedestrian" speed="13.89" length="159.15" width="2.00" shape="26.77,-166.79 12.36,-8.30"/>
<lane id="SC_1" index="1" disallow="pedestrian" speed="13.89" length="159.15" shape="29.36,-166.55 14.95,-8.06"/>
<lane id="SC_2" index="2" disallow="pedestrian" speed="13.89" length="159.15" shape="32.55,-166.26 18.14,-7.77"/>
</edge>
<edge id="WC" from="W" to="Node2" priority="-1">
<lane id="WC_0" index="0" allow="pedestrian" speed="13.89" length="197.95" width="2.00" shape="-193.59,9.17 4.35,11.11"/>
<lane id="WC_1" index="1" disallow="pedestrian" speed="13.89" length="197.95" shape="-193.57,6.57 4.38,8.51"/>
<lane id="WC_2" index="2" disallow="pedestrian" speed="13.89" length="197.95" shape="-193.54,3.37 4.41,5.31"/>
</edge>

<tlLogic id="Node2" type="static" programID="0" offset="0">
<phase duration="42" state="rrrrGGGgrrrrGGGgr"/>
<phase duration="3" state="rrrryyyyrrrryyyyr"/>
<phase duration="37" state="gGGgrrrrGGGgrrrrG"/>
<phase duration="5" state="gGGgrrrrGGGgrrrrr"/>
<phase duration="3" state="yyyyrrrryyyyrrrrr"/>
</tlLogic>

<junction id="E" type="dead_end" x="262.35" y="-4.62" incLanes="CE_0 CE_1 CE_2" intLanes="" shape="262.35,-4.62 262.64,3.77 262.35,-4.62"/>
<junction id="N" type="dead_end" x="-62.62" y="207.57" incLanes="CN_0 CN_1 CN_2" intLanes="" shape="-62.62,207.57 -70.42,204.46 -62.62,207.57"/>
<junction id="Node2" type="traffic_light" x="18.69" y="3.85" incLanes="SC_0 SC_1 SC_2 EC_0 EC_1 EC_2 NC_0 NC_1 NC_2 WC_0 WC_1 WC_2 :Node2_w0_0" intLanes=":Node2_16_0 :Node2_1_0 :Node2_1_1 :Node2_17_0 :Node2_4_0 :Node2_5_0 :Node2_5_1 :Node2_18_0 :Node2_8_0 :Node2_9_0 :Node2_9_1 :Node2_19_0 :Node2_12_0 :Node2_13_0 :Node2_13_1

:Node2_20_0 :Node2_c0_0" shape="11.37,-8.39 28.10,-6.87 28.22,-5.77 28.46,-5.39 28.83,-5.13 29.31,-4.98 29.92,-4.95 30.50,11.84 26.78,12.87 25.16,14.04 23.68,15.66 22.37,17.72 21.20,20.21 5.60,13.99 5.79,12.95 5.66,12.59 5.37,12.33 4.93,12.17 4.34,12.11 4.51,-4.69 6.51,-4.77 8.17,-5.07 9.48,-5.58 10.45,-6.31 11.08,-7.24">
<request index="0" response="10000000001100000" foes="10000000001100000" cont="1"/>
<request index="1" response="01111000011100000" foes="01111100011100000" cont="0"/>
<request index="2" response="01111000011100000" foes="01111100011100000" cont="0"/>
<request index="3" response="01110011011100000" foes="01110011011100000" cont="1"/>
<request index="4" response="00000000000000000" foes="00000011000000000" cont="0"/>
<request index="5" response="10000100000001001" foes="11000111000001111" cont="0"/>
<request index="6" response="10000100000001001" foes="11000111000001111" cont="0"/>
<request index="7" response="00110100000001000" foes="00110111000001110" cont="1"/>
<request index="8" response="00110000000000000" foes="00110000000000000" cont="0"/>
<request index="9" response="01110000011110000" foes="01110000011111000" cont="0"/>
<request index="10" response="01110000011110000" foes="01110000011111000" cont="0"/>
<request index="11" response="11110000011100110" foes="11110000011100110" cont="1"/>
<request index="12" response="10000000000000000" foes="10000000000000110" cont="0"/>
<request index="13" response="10000100000001000" foes="10000111110001110" cont="0"/>
<request index="14" response="10000100000001000" foes="10000111110001110" cont="0"/>
<request index="15" response="10000100001101000" foes="10000111001101110" cont="1"/>
<request index="16" response="00000000000000000" foes="01111100001100001" cont="0"/>
</junction>
<junction id="S" type="dead_end" x="34.14" y="-166.12" incLanes="CS_0 CS_1 CS_2" intLanes="" shape="34.14,-166.12 42.51,-165.36 34.14,-166.12"/>
<junction id="W" type="dead_end" x="-193.52" y="1.77" incLanes="CW_0 CW_1 CW_2" intLanes="" shape="-193.52,1.77 -193.44,-6.63 -193.52,1.77"/>

<junction id=":Node2_16_0" type="internal" x="8.47" y="-1.46" incLanes=":Node2_0_0" intLanes=":Node2_5_0 :Node2_5_1 :Node2_11_0 :Node2_c0_0"/>
<junction id=":Node2_17_0" type="internal" x="18.38" y="-3.30" incLanes=":Node2_3_0 NC_1 NC_2" intLanes=":Node2_5_0 :Node2_5_1 :Node2_7_0 :Node2_8_0 :Node2_9_0 :Node2_9_1 :Node2_13_0 :Node2_13_1 :Node2_15_0"/>
<junction id=":Node2_18_0" type="internal" x="26.07" y="2.60" incLanes=":Node2_7_0 WC_1 WC_2" intLanes=":Node2_1_0 :Node2_1_1 :Node2_3_0 :Node2_9_0 :Node2_9_1 :Node2_11_0 :Node2_12_0 :Node2_13_0 :Node2_13_1"/>
<junction id=":Node2_19_0" type="internal" x="16.10" y="12.74" incLanes=":Node2_11_0 SC_1 SC_2" intLanes=":Node2_0_0 :Node2_1_0 :Node2_1_1 :Node2_5_0 :Node2_5_1 :Node2_7_0 :Node2_13_0 :Node2_13_1 :Node2_15_0 :Node2_c0_0"/>
<junction id=":Node2_20_0" type="internal" x="11.37" y="4.57" incLanes=":Node2_15_0 EC_1 EC_2" intLanes=":Node2_1_0 :Node2_1_1 :Node2_3_0 :Node2_4_0 :Node2_5_0 :Node2_5_1 :Node2_9_0 :Node2_9_1 :Node2_11_0 :Node2_c0_0"/>

<connection from="EC" to="CS" fromLane="1" toLane="1" via=":Node2_4_0" tl="Node2" linkIndex="4" dir="l" state="O"/>

<connection from="EC" to="CW" fromLane="1" toLane="1" via=":Node2_5_0" tl="Node2" linkIndex="5" dir="s" state="o"/>

<connection from="EC" to="CW" fromLane="2" toLane="2" via=":Node2_5_1" tl="Node2" linkIndex="6" dir="s" state="o"/>

<connection from="EC" to="CN" fromLane="2" toLane="2" via=":Node2_7_0" tl="Node2" linkIndex="7" dir="r" state="o"/>

<connection from="NC" to="CE" fromLane="1" toLane="1" via=":Node2_8_0" tl="Node2" linkIndex="8" dir="l" state="o"/>

<connection from="NC" to="CS" fromLane="1" toLane="1" via=":Node2_9_0" tl="Node2" linkIndex="9" dir="s" state="o"/>

<connection from="NC" to="CS" fromLane="2" toLane="2" via=":Node2_9_1" tl="Node2" linkIndex="10" dir="s" state="o"/>

<connection from="NC" to="CW" fromLane="2" toLane="2" via=":Node2_11_0" tl="Node2" linkIndex="11" dir="r" state="o"/>

<connection from="SC" to="CW" fromLane="1" toLane="1" via=":Node2_0_0" tl="Node2" linkIndex="0" dir="l" state="o"/>

<connection from="SC" to="CN" fromLane="1" toLane="1" via=":Node2_1_0" tl="Node2" linkIndex="1" dir="s" state="o"/>

<connection from="SC" to="CN" fromLane="2" toLane="2" via=":Node2_1_1" tl="Node2" linkIndex="2" dir="s" state="o"/>

<connection from="SC" to="CE" fromLane="2" toLane="2" via=":Node2_3_0" tl="Node2" linkIndex="3" dir="r" state="o"/>

<connection from="WC" to="CN" fromLane="1" toLane="1" via=":Node2_12_0" tl="Node2" linkIndex="12" dir="l" state="o"/>

<connection from="WC" to="CE" fromLane="1" toLane="1" via=":Node2_13_0" tl="Node2" linkIndex="13" dir="s" state="o"/>

<connection from="WC" to="CE" fromLane="2" toLane="2" via=":Node2_13_1" tl="Node2" linkIndex="14" dir="s" state="o"/>

<connection from="WC" to="CS" fromLane="2" toLane="2" via=":Node2_15_0" tl="Node2" linkIndex="15" dir="r" state="o"/>

<connection from=":Node2_0" to="CW" fromLane="0" toLane="1" via=":Node2_16_0" dir="l" state="m"/>

<connection from=":Node2_16" to="CW" fromLane="0" toLane="1" dir="l" state="m"/>

<connection from=":Node2_1" to="CN" fromLane="0" toLane="1" dir="s" state="M"/>

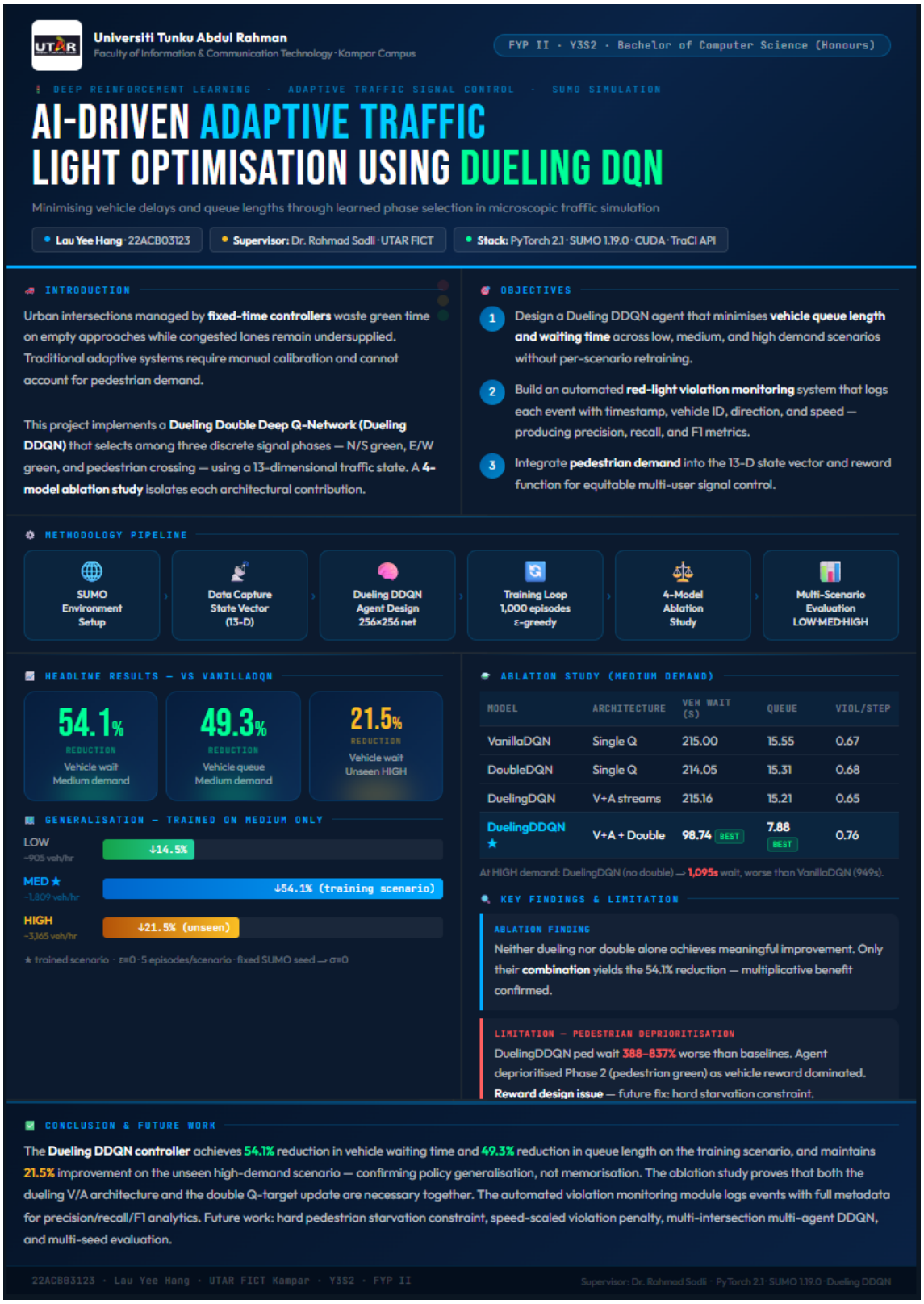
<connection from=":Node2_1" to="CN" fromLane="1" toLane="2" dir="s" state="M"/>

<connection from=":Node2_3" to="CE" fromLane="0" toLane="2" via=":Node2_17_0" dir="r" state="m"/>

<connection from=":Node2_17" to="CE" fromLane="0" toLane="2" dir="r" state="M"/>

```
    <connection from=":Node2_4" to="CS" fromLane="0" toLane="1" dir="l" state="M"/>
    <connection from=":Node2_5" to="CW" fromLane="0" toLane="1" dir="s" state="M"/>
    <connection from=":Node2_5" to="CW" fromLane="1" toLane="2" dir="s" state="M"/>
    <connection from=":Node2_7" to="CN" fromLane="0" toLane="2" via=":Node2_18_0" dir="r"
state="m"/>
    <connection from=":Node2_18" to="CN" fromLane="0" toLane="2" dir="r" state="M"/>
    <connection from=":Node2_8" to="CE" fromLane="0" toLane="1" dir="l" state="M"/>
    <connection from=":Node2_9" to="CS" fromLane="0" toLane="1" dir="s" state="M"/>
    <connection from=":Node2_9" to="CS" fromLane="1" toLane="2" dir="s" state="M"/>
    <connection from=":Node2_11" to="CW" fromLane="0" toLane="2" via=":Node2_19_0" dir="r"
state="m"/>
    <connection from=":Node2_19" to="CW" fromLane="0" toLane="2" dir="r" state="m"/>
    <connection from=":Node2_12" to="CN" fromLane="0" toLane="1" dir="l" state="M"/>
    <connection from=":Node2_13" to="CE" fromLane="0" toLane="1" dir="s" state="M"/>
    <connection from=":Node2_13" to="CE" fromLane="1" toLane="2" dir="s" state="M"/>
    <connection from=":Node2_15" to="CS" fromLane="0" toLane="2" via=":Node2_20_0" dir="r"
state="m"/>
    <connection from=":Node2_20" to="CS" fromLane="0" toLane="2" dir="r" state="M"/>
    <connection from=":E_w0" to="EC" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="CE" to=":E_w0" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":N_w0" to="NC" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="CN" to=":N_w0" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":Node2_c0" to=":Node2_w3" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":Node2_w0" to=":Node2_c0" fromLane="0" toLane="0" tl="Node2"
linkIndex="16" dir="s" state="M"/>
    <connection from=":Node2_w0" to="CW" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="SC" to=":Node2_w0" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":Node2_w1" to="CS" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="EC" to=":Node2_w1" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":Node2_w2" to="CE" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="NC" to=":Node2_w2" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":Node2_w3" to="CN" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="WC" to=":Node2_w3" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":S_w0" to="SC" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="CS" to=":S_w0" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from=":W_w0" to="WC" fromLane="0" toLane="0" dir="s" state="M"/>
    <connection from="CW" to=":W_w0" fromLane="0" toLane="0" dir="s" state="M"/>
</net>
```

Poster


Universiti Tunku Abdul Rahman
Faculty of Information & Communication Technology · Kampar Campus
FYP II · Y3S2 · Bachelor of Computer Science (Honours)
DEEP REINFORCEMENT LEARNING · ADAPTIVE TRAFFIC SIGNAL CONTROL · SUMO SIMULATION
AI-DRIVEN ADAPTIVE TRAFFIC LIGHT OPTIMISATION USING DUELING DQN
Minimising vehicle delays and queue lengths through learned phase selection in microscopic traffic simulation
Lau Yee Hang · 22ACB03123
Supervisor: Dr. Rahmad Sadli · UTAR FICT
Stack: PyTorch 2.1 · SUMO 1.19.0 · CUDA · TraCI API
INTRODUCTION
Urban intersections managed by fixed-time controllers waste green time on empty approaches while congested lanes remain undersupplied. Traditional adaptive systems require manual calibration and cannot account for pedestrian demand.
This project implements a Dueling Double Deep Q-Network (Dueling DDQN) that selects among three discrete signal phases — N/S green, E/W green, and pedestrian crossing — using a 13-dimensional traffic state. A 4-model ablation study isolates each architectural contribution.
OBJECTIVES
1 Design a Dueling DDQN agent that minimises vehicle queue length and waiting time across low, medium, and high demand scenarios without per-scenario retraining.
2 Build an automated red-light violation monitoring system that logs each event with timestamp, vehicle ID, direction, and speed — producing precision, recall, and F1 metrics.
3 Integrate pedestrian demand into the 13-D state vector and reward function for equitable multi-user signal control.
METHODOLOGY PIPELINE
SUMO Environment Setup
Data Capture State Vector (13-D)
Dueling DDQN Agent Design 256×256 net
Training Loop 1,000 episodes ε-greedy
4-Model Ablation Study
Multi-Scenario Evaluation LOW·MED·HIGH
HEADLINE RESULTS — VS VANILLADQN
54.1% REDUCTION Vehicle wait Medium demand
49.3% REDUCTION Vehicle queue Medium demand
21.5% REDUCTION Vehicle wait Unseen HIGH
GENERALISATION — TRAINED ON MEDIUM ONLY
LOW ~905 veh/hr ↓14.5%
MED ★ ~1,809 veh/hr ↓54.1% (training scenario)
HIGH ~3,165 veh/hr ↓21.5% (unseen)
★ trained scenario · ε=0 · 5 episodes/scenario · fixed SUMO seed → σ=0
ABLATION STUDY (MEDIUM DEMAND)
MODEL | ARCHITECTURE | VEH WAIT (S) | QUEUE | VIOL/STEP
VanillaDQN | Single Q | 215.00 | 15.55 | 0.67
DoubleDQN | Single Q | 214.05 | 15.31 | 0.68
DuelingDQN | V+A streams | 215.16 | 15.21 | 0.65
DuelingDDQN ★ | V+A + Double | 98.74 BEST | 7.88 BEST | 0.76
At HIGH demand: DuelingDQN (no double) → 1,095s wait, worse than VanillaDQN (949s).
KEY FINDINGS & LIMITATION
ABLATION FINDING
Neither dueling nor double alone achieves meaningful improvement. Only their combination yields the 54.1% reduction — multiplicative benefit confirmed.
LIMITATION — PEDESTRIAN DEPRIORITISATION
DuelingDDQN ped wait 388–837% worse than baselines. Agent deprioritised Phase 2 (pedestrian green) as vehicle reward dominated. Reward design issue — future fix: hard starvation constraint.
CONCLUSION & FUTURE WORK
The Dueling DDQN controller achieves 54.1% reduction in vehicle waiting time and 49.3% reduction in queue length on the training scenario, and maintains 21.5% improvement on the unseen high-demand scenario — confirming policy generalisation, not memorisation. The ablation study proves that both the dueling V/A architecture and the double Q-target update are necessary together. The automated violation monitoring module logs events with full metadata for precision/recall/F1 analytics. Future work: hard pedestrian starvation constraint, speed-scaled violation penalty, multi-intersection multi-agent DDQN, and multi-seed evaluation.
22ACB03123 · Lau Yee Hang · UTAR FICT Kampar · Y3S2 · FYP II
Supervisor: Dr. Rahmad Sadli · PyTorch 2.1 · SUMO 1.19.0 · Dueling DDQN