MSc Applied AI and Data Science

2022

Kok Leong LAU

Reinforcement learning for multi-intersection traffic light controls with lane area detection range

SOLENT UNIVERSITY
FACULTY OF BUSINESS LAW AND DIGITAL TECHNOLOGIES

MSc Applied AI and Data Science
Academic Year 2021–2022

K. L. LAU

Reinforcement learning for multi-intersection traffic light controls with lane area detection range

Supervisor: Dr Nick Whitelegg

September 2022

This report is submitted in partial fulfilment of the requirements of Solent University for the degree of MSc Artificial Intelligence and Data Science

Acknowledgements

I would also like to thank Dr Shakeel Ahmad, Dr Drishty Sobnath, and Prins Butt for their teaching and guidance over the course of my degree. I would like to especially thank Dr Nick Whitelegg and Dr Olufemi Isiaq for their continuous support and dedication to my research.

Abstract

Traffic congestion has always been a major issue in urban environments and is a huge contributor to carbon dioxide emissions and pollution. Solving this issue could reduce such pollutants as well as increase the productivity of the general population, reduce accidents, and bring down costs to resolve traffic-related problems. With the rise in artificial intelligence (AI) and deep Q-learning (DQN). More methods are becoming increasingly available for researchers to determine a data-driven approach to managing intersections controlled by traffic lights. Using tools like the simulation of urban mobility (SUMO), much research has been conducted into the suitability of using a reinforcement learning (RL) approach to control traffic light timings and traffic flow. Though positive outcomes have been reported, there seems to be a lack of consideration for the use of DQNs in a real-world scenario with the additional constraint of traffic cameras. This study aimed to implement a system that is adaptive to a real-world road infrastructure gathered from OpenStreetMap (OSM). Additionally, the impact of including lane area detectors with limited view distance to monitor queue length was also considered. The methodology includes a comparison of using a DQN with an increasing amount of hidden layers named MLP-0, MLP-1, and MLP-2 with the addition of a baseline fixed-timing (FT) approach. For each, results were gathered from 50 episodes consisting of 3,600 simulation time steps which is equivalent to 1 hour. Average time loss (ATL), average waiting time (AWT), vehicle count (VC), and episode rewards (ER) were the performance outcomes recorded and were logged using *tensorboard*. The results showed that MLP-2 reduced ATL and VC by 16.56% ($p < 0.05$) and 11.26% ($p < 0.05$) — respectively, whilst MLP-0 reduced AWT by 19.18% ($p < 0.05$). MLP-2 also had the largest episode rewards of 4084.75 amongst all the DQN models tested. To conclude and in comparison to the existing literature, it seems that placing an additional constraint with lane area detectors and using real-world data with minimal modifications — leads to a less performant outcome. It is the hope that this research can help guide future studies in considering real-world applications and scenarios for traffic light management systems.

Table of Contents

List of Figures

List of Tables

List of Abbreviations

AI	Artificial intelligence
ANN	Artificial neural network
ANOVA	Analysis of variance
API	Application programming interface
ATL	Average time loss
AWT	Average waiting time
CC BY-SA 2.0	Creative Commons Attribution-ShareAlike 2.0
CLI	Command-line interface
CNN	Convolutional neural network
CSV	Comma-separated value
DDPG	Deep deterministic policy gradient

DQN	Deep Q-learning network
DRL	Distributional reinforcement learning
ER	Episode rewards
FT	Fixed timing
GIS	Geographic information system
GPS	Global positioning system
GUI	Graphical user interface
IoT	Internet of things
LSTM	Long short-term memory
MDP	Markov decision process
ML	Machine learning
MLP	Multi-layer perceptron
ODbL	Open Data Commons Open Database Licence
OSM	OpenStreetMap
PICO	Population, intervention, comparison, outcome
PRISMA	Preferred Reporting Items for Systematic Reviews and Meta-Analyses
ReLU	Rectified linear unit
RL	Reinforcement learning
SUMO	Simulation of urban mobility
UCL	University College London
VC	Vehicle count

Introduction

(adapted from previous work, Lau 2022)

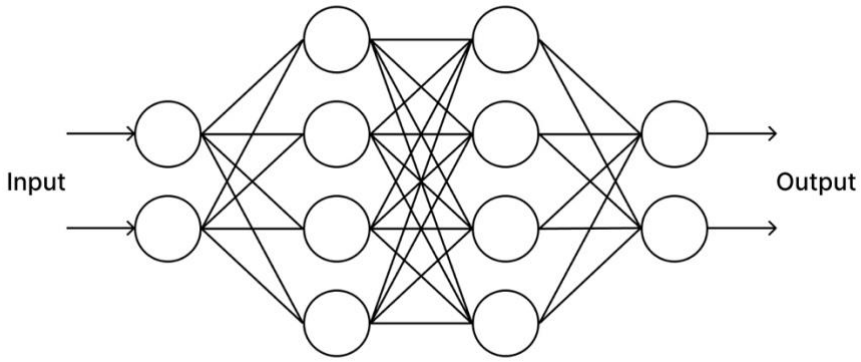
Through rapid urbanisation, traffic congestion has become a common issue in cities around the world. Before the outbreak of COVID-19, annual vehicle miles in the UK were steadily increasing — reaching a peak of 356.5 billion miles in 2019 (Department for Transport 2021). This presents a new challenge for the road and transport authorities who are overseeing the management of traffic flow to increase throughput and minimise incidents across the network.

With the rise of machine learning (ML) capabilities in recent times, many questions are being asked about the use and potential of artificial intelligence (AI) in critical infrastructures — like in this instance — transport. Much of the previous research seemed to agree that if some AI were to be implemented to solve congestion issues, it would typically be at the intersections controlled by traffic lights. Additionally, it is common to see reinforcement learning (RL) algorithms for these use cases due to the complexity and irregularities of the system that a typical artificial neural network (ANN) would not be able to solve (Sugiyama 2015).

Much research has been conducted in assessing the suitability of a deep Q-learning network (DQN) on road-traffic infrastructure and how it may be able to achieve greater throughput at intersections. However, there is a gap in bridging the theoretical and the implementation of such a system in the real-world. Updating or building infrastructure like road networks needs much planning, funding, and time. Hence, it is crucial to prove whether such a system would behave as expected in real-world applications. Some of the research has considered these scenarios, for instance, the multi-agent approach of using neighbouring intersections and importing real street maps to train the model (He *et al.* 2021, R *et al.* 2022). However, they've failed to recreate a situation where an agent has limited visibility — meaning vehicles that are within the range of an intersection can be considered. It may not be possible or feasible to know the exact location of all vehicles in a road network. The veracity of the data collected depends on the placements of cameras and internet of things (IoT) devices which usually can be conveniently placed at the intersections. Briefly, this project aims to train an effective DQN that considers the vehicle detection range of intersections in a larger network using lane area detectors. Doing so could help set the stage for future research and implementation of smart traffic signalling or management systems.

An ANN is made up of multiple layers of neurons which are inter-connected between each layer. There are also different types of layers where each would be more suited to a particular type of input or purpose. For instance, each neuron in a dense layer is connected to all preceding neurons in the previous layer "Figure 1". Its benefits are that it is structurally agnostic — meaning that no assumptions about the inputs are required. Also, as it is connected to all the neurons in the previous layer, it can learn from all the given features. However, a drawback is that it is computationally expensive due to the high number of parameters present in these layers (Guresen and Kayakutlu 2011). ANN is an umbrella term, and this is one such example of a multi-layer perceptron (MLP). Another type is a convolutional neural network (CNN) which is typically used for image processing, classification or any purposes where local spatial correlation is important — meaning that distances in your data matter (O'Shea and Nash 2015).

Figure 1 Example of an MLP with dense layers



To further understand the inner mechanics of an ANN, consider the following example of a perceptron which consists of a singular neuron, one output, has n number of inputs and is used in supervised machine learning "Figure 2". Supervised machine learning is the process of training a model with a set of input data and its known labels. For instance, training with a set of images of different types of vehicles and trying to map its prediction to correctly identify if it is a motorcycle, bus, van, etc. The inputs of a perceptron are key traits or features of the data that the model will learn from — represented as I1 to I4. For any prediction to be useful, some mathematical operations are performed on the inputs to emphasise the importance of certain features. This is achieved through finding the product of the input value and its weight (W1 to W4) which is randomly initialised. These adjusted values are then fed through an activation function which produces a transformed output which will determine if the neuron will be

activated if it surpasses a certain threshold. Like in a binary classifier where the threshold could be 0.5, a higher value would activate the neuron and give its respective input the corresponding classification (Rosenblatt 1958). Elaborating on activation functions, these are mathematical operations that take an input value and produce an output after passing it through a function like sigmoid, rectified linear unit (ReLU), or tanh “Figure 3”. The neural network learns through a method called backpropagation. Optimisers like *Adam* which is derived from the name adaptive moment estimation — and is a stochastic gradient descent algorithm used to update the network's weights iteratively in an attempt to improve the accuracy of its predictions (Kingma and Ba 2017).

Figure 2 Example of a perceptron

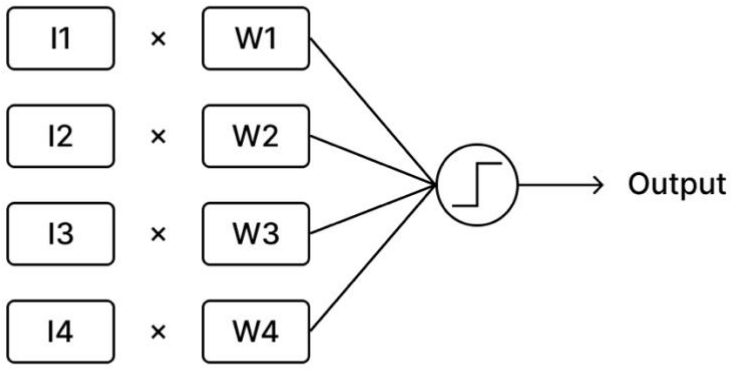
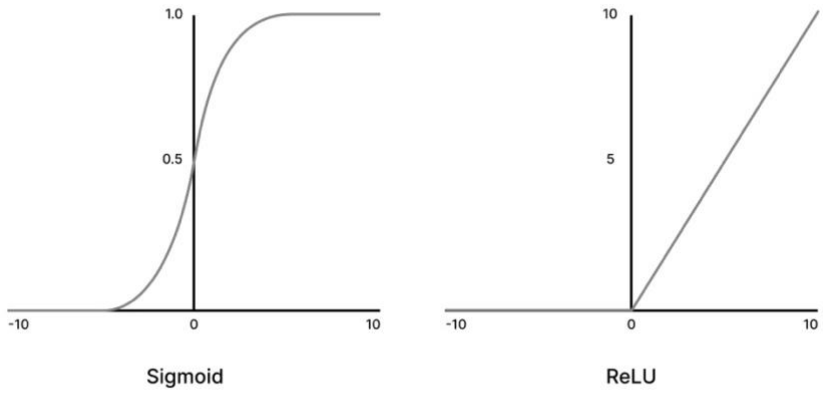
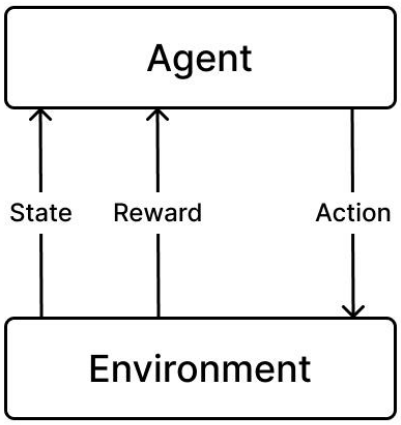


Figure 3 Sigmoid and ReLU activation functions



Whilst the mechanics of an ANN closely resembles the human brain, RL can be seen as an equivalent representation of learning through trial and error — much like training a dog. An agent, which can be seen as the 'Player' or 'Controller', is used to interface with a given environment and will learn which actions yield the largest reward before storing the experience for later recall "Figure 4" (Wells and Bednarz 2021).

Figure 4 Reinforcement learning concept



Another important concept in the field of RL is Q-learning. Simply, this method of learning utilises a Q-table to maintain a record of values that represents the most viable actions taken for a given state. Further elaborating, a state is a snapshot of the environment — for instance, a frame in a video game would be a state. The values inside a Q-table are also known as Q-values. The goal of this algorithm is to maximise the rewards gained from each step and subsequently improve the Q-values through training the agent with the environment over a set number of iterations — more commonly known as episodes. The environment will reset at the end of each episode but the data within the Q-table will persist over the course of training. Firstly, the agent will initialise and pick a random action to begin the training. After, it will assess the change in the environment and ascertain a reward relating to the effects of its action on the environment — or state-action. From this, it calculates the values in the Q-table for that state and repeats the above for the rest of the episode. From the second episode onwards, the agent would have populated the Q-table with the initial sets of values. Following onwards, it will now take the actions with the largest Q-value. Each episode will reinforce the agent's understanding of the environment as more successful actions for a given state — will yield higher Q-values over the course of the training "Table 1" (Fan *et al.* 2020).

Table 1 Example Q-table with 4 actions and N states (shaded are selected actions)

	Action 1	**Action 2**	**Action 3**	**Action 4**
State 1	0.11	0.25	0.73	0.29
State 2	0.64	0.83	0.13	0.36
State 3	0.56	0.89	0.68	0.53
State 4	0.64	0.54	0.17	0.18
State 5	0.53	0.12	0.76	0.10
State N	…	…	…	…

However, it is easy to see why a tabular approach with Q-learning could pose an issue for larger and more complex environments. Its simplicity is an advantage if there are a finite number of states, but it quickly becomes computationally expensive and inefficient when the duration is unknown. In such a scenario, a possible solution is to take an approach that approximates the Q-function or curve — meaning that it is viable to train the agent using ANNs to generate the Q-values. Though this resolves the issue, it raises numerous challenges and increases the complexity of the algorithm. For instance, the agent would need a replay memory which stores previous experiences including past states, the reward gathered, the action that was taken, and the following state. These experiences will be sampled by the agent for training. Additionally, the use of ANNs would require more hyperparameters and tuning to optimise the performance of the model — further increasing the complexity.

Background

(adapted from previous work, Lau 2022)

Understanding the current literature surrounding the topic of using DQNs in traffic light management systems required a systematic approach due to the vast number of studies existing in this field. The following was performed to increase the quality of the literature review — subsequently the accuracy and relevance of this research.

Finding articles

The research question and database search criteria “Table 2” were formulated following PICO (problem, intervention, comparison, and outcomes). In addition, the PRISMA framework was implemented to guide the background research and literature review. Initially, search results were obtained and exported as a comma-separated values (CSV) file which was imported into a resolution tool named *Rayyan*. Duplicates were detected and removed, followed by the first pass of exclusions through abstract reading; then, the second pass was by reading the full text.

This process gathered initially 945 but following the removal of duplications and exclusions via title or abstract — 74 articles remained. Finally, after full-text exclusions, the remaining count of articles was 53.

Table 2 Search criterion

Databases	IEEE Xplore
Search terms	"traffic" OR "traffic light" AND "congestion" OR "intersection*" OR "junction*" AND "reinforcement learning" OR "deep q-network" OR "dqn" OR "machine learning" OR "deep learning" AND "management*" OR "reduction" OR "control"
Free-text version	("traffic" OR "traffic light") AND ("congestion" OR "intersection*" OR "junction*") AND ("reinforcement learning" OR "deep q-network" OR "dqn" OR "machine learning" OR "deep learning") AND ("management*" OR "reduction" OR "control")

Inclusion and exclusion criterion

The PICO framework was used to identify articles that are relevant and within the scope of the project. From this, inclusion and exclusion criteria were created as per "Table 3" to further refine search results. No criteria for comparison were created to broaden the scope of the search due to uncertainty as to which comparators were used.

Table 3 PICO framework for inclusion/exclusion criterion

	Include	**Exclude**
Problem	Traffic management at nodes/junctions	Not traffic related
Intervention	Reinforcement and/or deep learning	-
Comparison	-	-
Outcomes	Reduce congestion, better flow, or less incidents	-

Current literature

From the included articles, it appears that much of the research has been conducted using Simulation of Urban Mobility (SUMO) for various questions regarding real-world data, single/multi-intersection problems, and optimisations in the algorithm (Liu *et al.* 2017, Wu, Kong, *et al.* 2020, Joo and Lim 2021). Also, common performance measurements across the

studies include the waiting time, throughput, queue length, carbon dioxide emission, noise pollution, and fuel consumption (Liu *et al.* 2017).

Much of the research also described and implemented the Markov Decision Process (MDP) which is a standard mathematical model for Q-learning and as discussed previously in the introduction (Rosyadi *et al.* 2016, Garg *et al.* 2018, Tan *et al.* 2020). Simply, the model represents the interaction of the agent depending on the state and reward from a given action performed in an environment. Hence, Q-learning can be expressed as below where s is the state, a is the action, α is the learning rate, γ is the discount factor, and R^t is the reward at the time t. The expression produces the Q-value at time t for the given state and action, $Q(s_t, a_t)$ is the value for the current state-action pair, and $max_{a'}Q(s', a')$ is the estimated maximum reward for the next action — whilst γ applies a discount to this value (Liu *et al.* 2017).

$$Q^{new}(s_t, a_t) = Q(s_t, a_t) + \alpha\left(R^t + \gamma max_{a'}Q(s', a') - Q(s_t, a_t)\right)$$

As discussed, DQNs approximate this function using ANNs. The input of a DQN will be a representation of the current state, like in this case, the number of vehicles on the road or at the intersection. The information then propagates forward through the ANN with the output layer corresponding with the action space where the agent will pick the action with the highest Q-value to perform (Chen and Kulla 2019). After taking the action, the agent will ascertain the reward of that particular state-action pair and the new state before storing it in the replay memory. After, the agent will sample the memory with the batch size which is a hyperparameter configured prior to training. This becomes the input for the ANN using backpropagation to adjust the weights between neurons to improve the accuracy of its future predictions.

It is also common to see 2 neural networks used for a DQN agent. They are both the same in structure but differ in purpose. Q-learning uses an exact value function that calculates the Q-value at every step the agent takes as previously discussed. Since DQNs are function approximators and the weights in the neural network are randomly initialised, their predictions can be wildly different and unstable. Additionally, the network is being trained on every single timestep — not allowing enough iterations for the adjustments to converge towards the target

value. Hence, introducing a second network named the ‘target network’ can help resolve this issue. Simply, it freezes the target value to allow the first ANN to converge towards it for N time steps before syncing the weights — after which the process repeats. Consider the following example in “Figure 5” and “Figure 6”. With the aid of the target network, the predicted next state values will not deviate substantially — allowing the other network to converge towards this value prior to syncing the weights between these two networks.

Figure 5 DQN failing to converge to target value with no target network

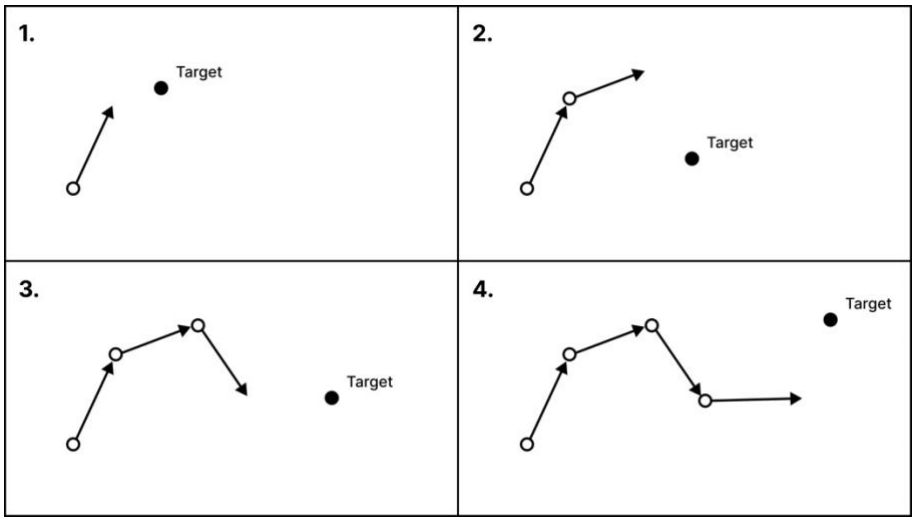
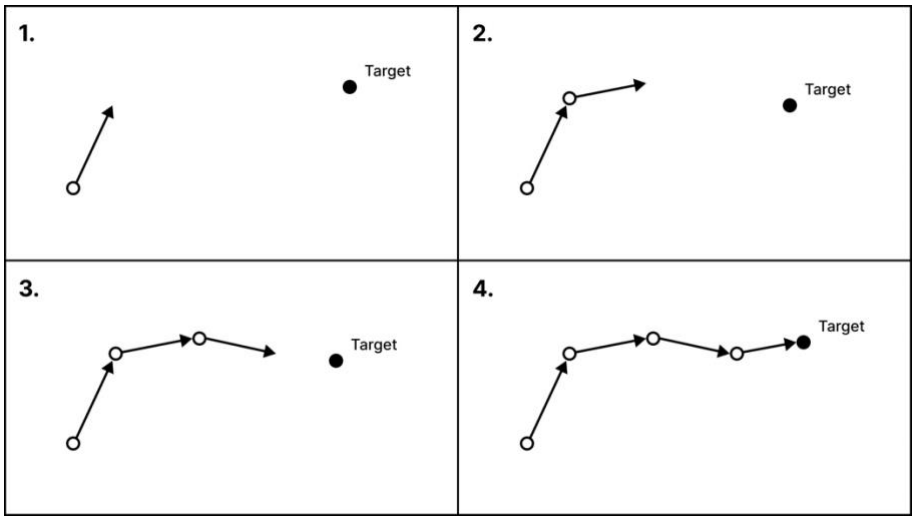


Figure 6 DQN converging to the target value using a target network

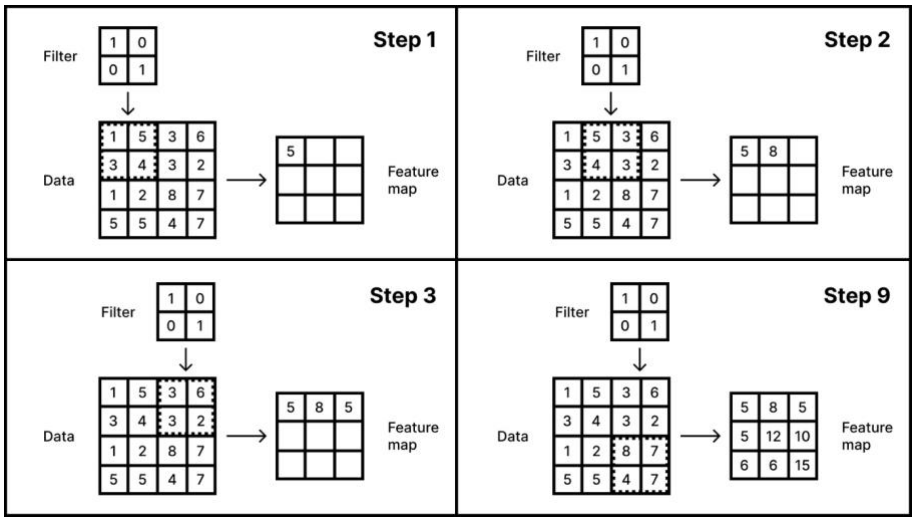


Though the problem statement of the included articles is the same — being to improve traffic flow at intersections. Many different approaches have been observed for optimising models and objective functions. For instance, one such case utilised a CNN with image data from a

custom simulation to predict to next best action. The model was basic and it consisted of multiple convolution layers followed by a fully connected output layer (Garg *et al.* 2018). Contrastingly, other authors have opted for a more rounded approach — considering multiple metrics for their agent. An example would be to combine the velocity and position matrices into a 2-channel image which will become the input for a CNN with ReLU activation functions. Additionally, queue lengths can be represented as a matrix of lanes and roads which again can be handled by a CNN. Pedestrians and traffic light phase vectors can be passed into a fully connected layer before being concatenated with all the results from the computations of the above states. Then, a long short-term memory (LSTM) network can finalise the combined states and formulate an action (Wu, Zhou, *et al.* 2020). A benefit of a more complex network is the holistic approach to the environment's states — considering multiple aspects of the simulation could add to the practical applications of the research.

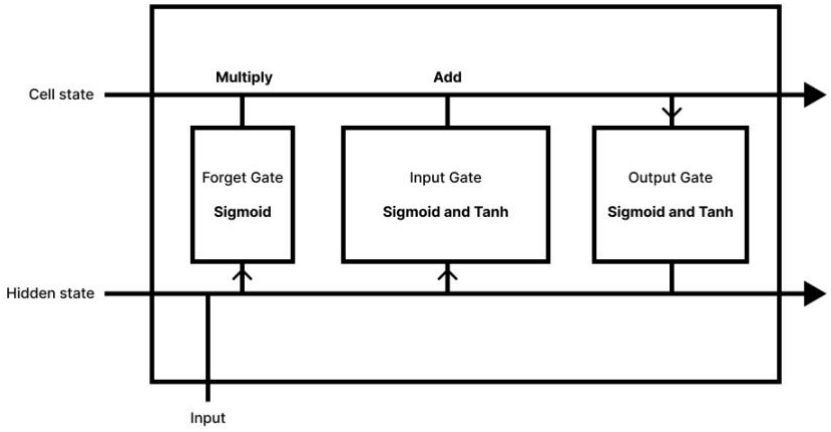
With regards to the other types of ANNs as per the above, each has its own benefits and drawbacks which ultimately determine their suitability for certain use cases. Starting with CNN, it is common to see these types of networks used for image-based learning due to their robustness in detecting unique features in 2-dimensional data. It is important to note that they are not limited only to images, but rather any 2-dimensional representation of features. For instance, the current position and velocity of cars on a road network can be represented as a matrix. Key components of convolutional layers in a CNN include the filter, data, and feature map. As per "Figure 7", the filter is defined with dimensions 2x2 and initialised with kernel values that will perform multiplication on the input. In the first step, the following was performed, $1(1) + 0(5) + 0(3) + 1(4)$ to attain a value of 5 on the feature map. After, the filter is shifted right by a parameter called *stride* which is 1 in this case. The following steps will repeat the same calculations above. The final step will gather a completed feature map of the input with reduced dimensions (O'Shea and Nash 2015). CNNs can have multiple convolution layers to create multiple feature maps where the model can detect important information in each layer. Some drawbacks of using a CNN include the requirement of large datasets to thoroughly improve the validity of the feature map. Furthermore, the orientation and position of the data cannot be encoded. Solutions to these issues exist such as data augmentation where a set of images/data is duplicated but transformed randomly in orientation and/or position to add variety and increase the amount of training data (Han *et al.* 2018).

Figure 7 Example of how convolution works



Shifting the focus to the mechanics of LSTMs, a high-level overview would describe 4 key components of an LSTM cell —the forget gate, input gate, output gate, and the cell state as per "Figure 8". Multiple cells are created sequentially where 2 sets of states are passed to connecting cells known as hidden states and cell states. Each gate contains an activation function like a sigmoid or tanh function. From the beginning, the previous cell's hidden state and the current cell's input are combined into a new hidden state. The forget gate decides how much of this new combined hidden state should be forgotten using a sigmoid function. A value tending towards 1 means that the cell remembers more and vice versa. The output of the forget gate is then multiplied by the cell state to adjust its value. After, the new hidden state passes through the input gate which transforms the value separately through a sigmoid and tanh function before multiplying the two together to produce the output for this gate. This output is then summed to the cell state. The final output gate takes the processed cell state and transforms it using a tanh function; additionally, it also takes the new hidden state and passes it through a sigmoid function — where both transformed values are multiplied together to create the hidden state for the following cell (Wang 2017).

Figure 8 States and gates of an LSTM cell

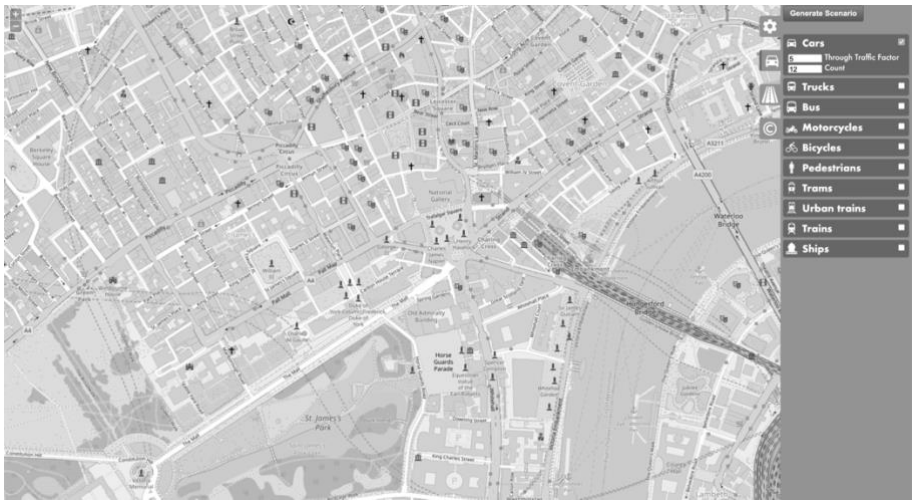


There is a split between the articles about the data or method adopted for the simulation. Some chose to use artificially created networks that are rigid in layout; for instance, a 3x3 grid of 4-way intersections (Shinde *et al.* 2021). This method simplifies the scenario and can assist in understanding the core principles behind a road network that is optimal and making computations simpler. However, the trade-off would be its reflection on real-world use. The contrary involves importing real map data into SUMO and running the simulation with pre-defined parameters like the number of vehicles and their type. This could be achieved by using one of SUMO's first-party tools called *osmWebWizard.py* which uses OpenStreetMap (OSM) in a custom graphical user interface (GUI) to crop out areas to generate the simulation network (Liu *et al.* 2017, OpenStreetMap 2022). An example of a multi-intersection network that is commonly used by authors consists of numerous nodes that are 4-way intersections. This is due to the notion that generally, the most throughput of traffic occurs at these larger intersections (R *et al.* 2022, Wang *et al.* 2022). Hence, they have a greater influence on the region's congestion rate which typically has more cameras and IoT devices to monitor its status — allowing for more data to be gathered.

OSM provides accurate geographical data for websites, apps, and software — which is community driven and built by local mappers, geographic information system (GIS) professionals, and engineers. It has many partners like University College London (UCL). Its purpose is to emphasise local knowledge where contributors submit global positioning system (GPS) data, aerial imagery, and low-tech field data relating to roads, trails, shops, stations, and more (OpenStreetMap 2022). Furthermore, OSM is registered under the Open Data Commons

Open Database Licence (ODbL) and their documentation is registered with the Creative Commons Attribution-ShareAlike 2.0 license (CC BY-SA 2.0). Hence, researchers and engineers can freely use their software as long as it is credited. OSM's existing purpose from the current pool of literature and for this project's use — is as a tool for SUMO to generate simulation data where the user can configure the traffic flow factors, the area of the road network to simulate, and the types of vehicles in the simulation "Figure 9".

Figure 9 Example of an OSM SUMO tool named osmWebWizard.py



Previous authors have outlined some key configurations for the hyperparameters used in DQNs for traffic control "Table 4". Starting with epsilon, this represents the starting value of the epsilon-greedy strategy for agents. Simply, it is a value that decays over the course of the episode — increasing the likelihood that the agent will exploit the current model and replay memory to take a non-random action (Dabney *et al.* 2020). Secondly, gamma is a discount factor for rewards. Simply, it constitutes how much the agent cares about rewards in the future compared to ones in the immediate term. It ranges between 0 and 1 where 0 indicates the agent only cares about immediate rewards and chooses the corresponding actions to achieve this (François-Lavet *et al.* 2016). Replay size and batch size are configurations for creating and interacting with the replay memory. Replay size defines the total number of experiences that can be stored in the agent's memory whilst the batch size is how many experiences to sample for training. Having a larger replay size can result in a more stable DQN but would require more computational resources.

Table 4 List of hyperparameters and their description

Epsilon	The value used in the epsilon greedy strategy.
Gamma	This is a discount factor for reward.
Replay size	The fixed size of the replay memory working in a queue data structure.
Batch size	The number of experiences to sample from the replay memory to train with.

As for parameters, different values of traffic density were observed among the included articles. Generalising a few, it is common to see classification to be graded as low, medium, or high levels of density where one such study reportedly used 10.67, 13.33, and 16 units per second — respectively (Pang and Gao 2019). Similarly, for the model's hyperparameters, many variations were gathered from the current pool of literature. Batch sizes ranged from 16 to 64, learning rates were between and around 0.0001 and 0.0005, whilst replay memories were around 10,000 to 20,000 (Liu *et al.* 2017, Pang and Gao 2019, Wu, Zhou, *et al.* 2020). The only common similarity amongst most articles was the discount rate of 0.99.

In general, most articles support the use of RL and DQN-based models to manage traffic signals at intersections over traditional fixed timing (FT) operations. Some reported results were decreased travel times and decreased average stop time during peak hours of 38.97% and 62.38% — respectively (Ge 2020). Other studies have also considered additional constraints that could affect traffic flow at intersections like pedestrians. One such article used data from neighbouring nodes as well as its own to train an agent — resulting in an improvement of 16.7% for waiting time during high pedestrian scenarios compared to other multi-agent approaches (Liu *et al.* 2017). In contrast, some authors indicated that though DQN is an effective method, it may not be the best. For instance, a deep deterministic policy gradient (DDPG) was shown to have more improvements. Simply and without going into detail as that would exceed the scope of this research, it consists of a critic model which predicts the Q-value and an actor model which determines which action to take. It was shown that under low, medium, and high traffic density scenarios, the average delay time for DDPG was 20%, 17%, and 10% lower compared with a DQN alternative — respectively (Pang and Gao 2019).

Some studies have also identified the limitations of using SUMO and DQNs. A common practice to transform data into observation spaces will be to represent the network as a matrix. The values within could represent the occupancy of a vehicle, its velocity, or any attributes associated (Pang and Gao 2019, Wu, Zhou, *et al.* 2020, Shinde *et al.* 2021). Hence, more computing resources will be needed for training larger networks. Regarding SUMO, not all

scenarios are realistically represented — especially collisions. The default setting for handling collisions is to teleport the vehicles to the next edge on its route with additional settings to instead throw a warning, do nothing, or remove both vehicles from the simulation (Garg *et al.* 2018, SUMO 2022).

Methods

Key objectives include building an RL model that interacts with SUMO using its traffic control interface (TraCI) API. Additionally, the simulation was performed using data from OSM to emulate existing road networks. Taking a different approach to some current literature, intersections in this project were isolated — meaning that it did not consider the states of neighbouring intersections. The rationale for this was two-fold. Firstly, this reduced the overall simulation complexity and training of the models. Secondly, being independent of a larger network could potentially generalise the model in a scenario if one or more neighbouring nodes were broken. Hence, the intersection takes the best action for itself regardless of the surrounding intersections. Another benefit to this method was that if it were to be deployed in a production environment, the model behaves continuously rather than in a localised manner. Simply, the model will not be specific to a certain area — unlike in the case where it has been trained using a region of a global network (Liu *et al.* 2017).

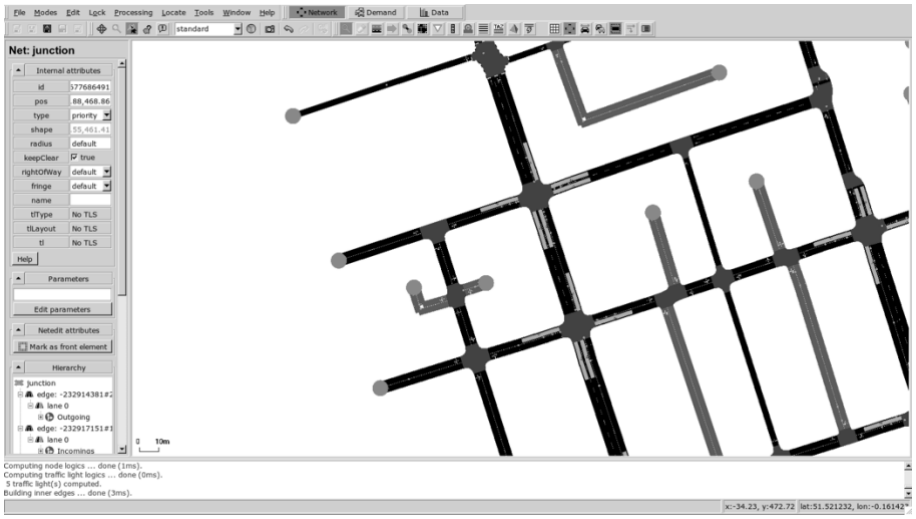
The methodology of the experiment will include a comparison of key performance metrics for different variants of the ANN implemented for the agent. This has deviated from the original project plan which originally was interested in the performance of the agent at different traffic flow rates. The rationale for this change was that the traffic flow used was based on real data gathered from local authorities — with the project also taking a different approach to multi-intersection problems. Hence, it was more beneficial to assess the suitability of this DQN and its implementation with different layers and structures. These results were also compared to a control situation — the FT approach for traffic lights where it used its default intervals gathered from OSM.

The hypothesis for this project expected a negative outcome of the artefact across the metrics in comparison to existing research; however, a positive effect was anticipated when compared to the traditional FT approach for traffic lights. The belief behind this was due to the additional constraints placed on the simulation with reduced detector ranges. Though no human-related data was required — ethical clearance was also attained “Appendix A”.

State

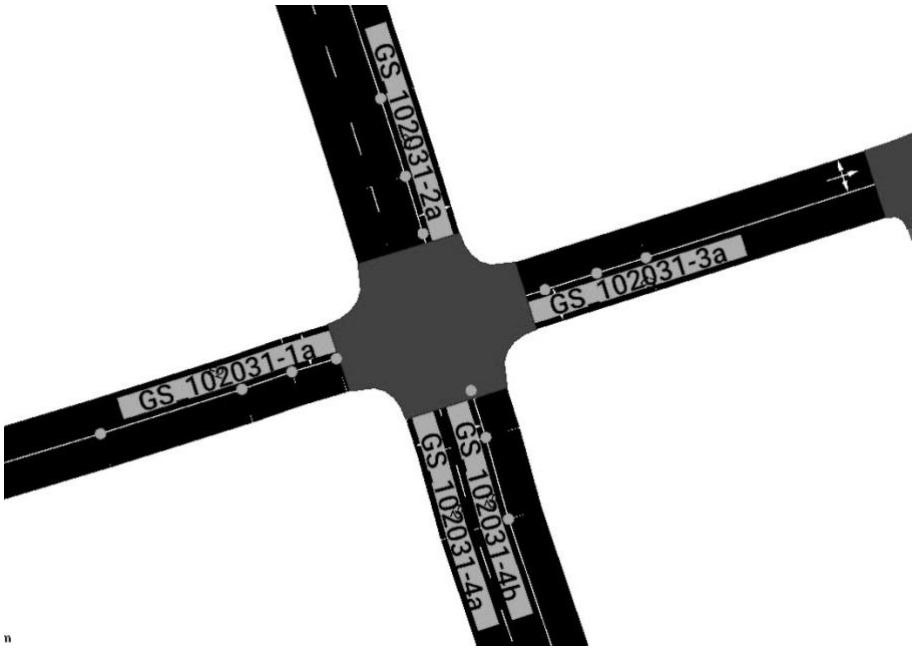
Lane area detectors were placed at every controlled intersection in the incoming direction using SUMO's first-party tool *netedit* "Figure 10" which is a GUI for creating or editing a road network. These lane area detectors were either 15 or 20 meters depending on the length of the road.

Figure 10 SUMO's netedit tool



Each detector was named using the convention where the intersection name was joined with a lane identifier. This identifier described the direction and position of the lane with respect to its position in the intersection. Working from the most westerly incoming direction, each road (or 'edge') was labelled with a number — incrementing as it rotates clockwise about the intersection. Simply, westerly roads were labelled as 1, northernly as 2, easterly as 3, and southernly as 4. Similarly, for each lane within a road, it was given a sub-label denoting its position starting from the left-most lane "Figure 11". For example, the second lane of the most northernly incoming road for intersection GS0001 will have an identifier of GS001-2b.

Figure 11 Example of lane area detector identifier convention



Using these identifiers, the artefact can observe the current state of the environment by using the TraCI API. This returns a count of all vehicles in the given range of each detector. The total queue length of the intersection is simply the sum of all detector counts. The state used for the agent is also represented as a vector of the queue length for each lane. For example, 5 lane area detectors will have a vector that contains 5 values representing the queue length.

Action

To retain the realism of the simulated road network, the original number of active phases the intersections have were not changed. Instead, as discussed later, each agent will dynamically instantiate ANNs with the correct number input and output space that are unique to that intersection. Unlike the states, implementing actions in the agent was much simpler as all cases can be handled directly from the TraCI API. As the agent was initiated, it looks for the traffic light programme as specified in the simulation configurations. It will return numerous phases with attributes like phase duration. The agent simply counts the number of phases available which formed the number of actions that the agent can take.

However, there is a limitation with this project regarding its realism. To reduce the complexity and to keep the study within the scope of the research question, all yellow phases of traffic lights were removed. This was common amongst the current literature as considering these

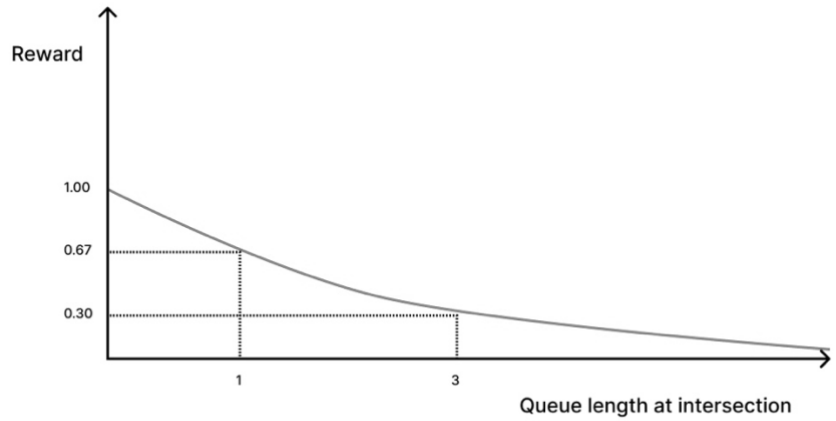
intermediate phases would place additional requirements in the artefact's implementation like realising a phase queuing process as these yellow phases would need to exist in between each active phase. For instance, before a light turns red, it will need to show the yellow phase — causing issues if the agent decides to execute a different phase during this time.

Additionally, there is a minimum active duration for all phases as the agent will retain the selected action for at least 10 steps. The rationale for this is to prevent unrealistic behaviour where the agent would constantly switch phases between steps. In reality, drivers would need more than just a few seconds to react accordingly to the changes in the traffic light phase and need time to pass through the intersection.

Reward

Designing an expression to represent the reward for the agent at each time step required some consideration of the current state at the intersections. Nonetheless, a simple solution was proposed where the objective of the rewards function is to return positive feedback for an action that reduced the total number of vehicles queued at an intersection. For when there are no vehicles queued, the agent should receive a maximum reward. The alternative is for when the number of vehicles tends to a higher count, the reward should decrease exponentially to quickly discourage extremely inefficient actions. Having a reward between 0 and 1 achieved this where 1 represented the largest reward possible and 0 indicated the action was extremely detrimental. In the end, the curve $f(x) = 1.5^{-x}$ was created through trail-and-error with the base value whilst keeping the negative exponent to represent an inverse relationship with total queue length "Figure 12".

Figure 12 Reward function



Simulation

As discussed, the tool commonly used for traffic simulation is SUMO. It is open-sourced with regular maintainers. It comes with many tools to help researchers and engineers configure a simulation environment that is tailored to their needs. Some of these tools have been discussed already; however, one that is worth mentioning would be the *randomTrips.py* helper module. This module generates a specified amount of random journeys and the number of vehicles to populate the simulation with over a given time period.

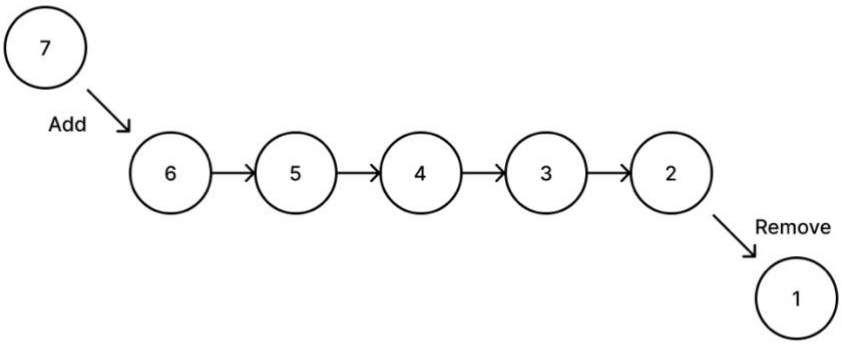
In this project, the maximum simulation time is 3600 steps which correspond to 1 hour in seconds. The geographical location considered for this study was London around the coordinates 51.52040° N, 0.15955° W due to its high tendency of traffic and variety of roads and intersections. Using historical data from the *Depart for Transport*, the most recent daily motor traffic count in this area was 11,821 (Department for Transport 2021). Using this and adjusting with the assumption that this value mostly covers the working hours, 1,478 trips per hour formed the traffic flow rate used in the simulation.

Replay memory

A key requirement for memory is that it is limited and for older experiences to be pushed further back in their position until it is forgotten “Figure 13”. This can be achieved using a deque from the standard library *collections* — whilst organising key methods of the replay memory in a class “Appendix B”. Upon initialising the memory, a deque is created using the specified size. Additionally, the length of the class instance is represented as the length of the deque using a

'magic method'. There are 2 public methods that can be called from this class. The *push* method adds a new experience to the deque in the form of a named tuple "Appendix C". The latter method is used to sample the current collection in the memory with the specified batch size. For example, if 32 was the batch size, then that amount of experiences will be stochastically drawn from the current collection.

Figure 13 Example replay memory with a size of 5



Neural Networks

Since the aim of this project is to consider a traffic scenario with the additional constraint of lane area detectors to count the number of vehicles at the intersections, the state can simply be represented as a vector of size N where each item represents the lane with queue length l — as previously introduced. This corresponds with the input layer of the ANN where the size will depend on the number of lane detectors situated at the traffic light intersection.

Since the network only considers the above and should take an action based on the immediate state of the environment, an MLP was implemented due to its simplicity and needing less configuration. This decision was also made on the premise of what the literature revealed regarding the use of LSTMs in traffic light DQN experiments. Nonetheless, multiple MLPs were created to determine whether the number of layers and the neurons within would impact the DQN agent's ability to manage traffic. It was decided that 3 variants of MLP would be tested subsequently named MLP-0, MLP-1, and MLP-2 where each respective variation has increasingly more hidden layers that generalise the data as it propagates forward through the

network. Between each layer, a ReLU activation function was used due to its simple but efficient nature where negative values are set to 0 before increasing linearly — which reduces the computation time (Hara *et al.* 2015). MLP-0 has 1 hidden layer of 128 neurons before the output layer "Figure 14". MLP-1 has 2 layers of 128 and 64 neurons respectively "Figure 15". MLP-2 has 3 layers with 128, 64, and 32 neurons in each layer respectively "Figure 16".

Figure 14 Example of MLP-0

```
Sequential(
  (0): Linear(in_features=5, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=2, bias=True)
)
```

Figure 15 Example of MLP-1

```
Sequential(
  (0): Linear(in_features=5, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=2, bias=True)
)
```

Figure 16 Example of MLP-2

```
Sequential(
  (0): Linear(in_features=5, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=32, bias=True)
  (5): ReLU()
  (6): Linear(in_features=32, out_features=2, bias=True)
)
```

As previously eluded to, the agent consists of 2 ANNs — one of which is the target network. "Appendix D" demonstrates the implementation of this crucial flow in the project. Firstly, the agent samples from the replay memory a collection of past experiences before processing it into batches grouped by states, actions, rewards, and next states. Using the states, the first ANN was used to generate some predicted Q-values and only select ones that correspond with the action that was taken using the *gather* method. For clarity, let us call these predicted Q-values Q1. Next, the same is executed for the target network but instead with the next state values as

the input and selecting the largest Q-values that correspond with each state — calling it Q2. Further transformations were performed on Q2 to apply the discount rate γ and the rewards associated. Finally, the loss was calculated between Q1 and Q2 before executing backpropagation to update the weights of the first ANN.

Main loop

A custom command-line interface (CLI) was created to run the source code using *argsparse* with default values "Appendix E". This allowed easy configuration when specifying hyperparameters for the agent, neural networks, or the simulation environment. Upon executing the *main.py* file, these configurations are logged to the terminal for confirmation.

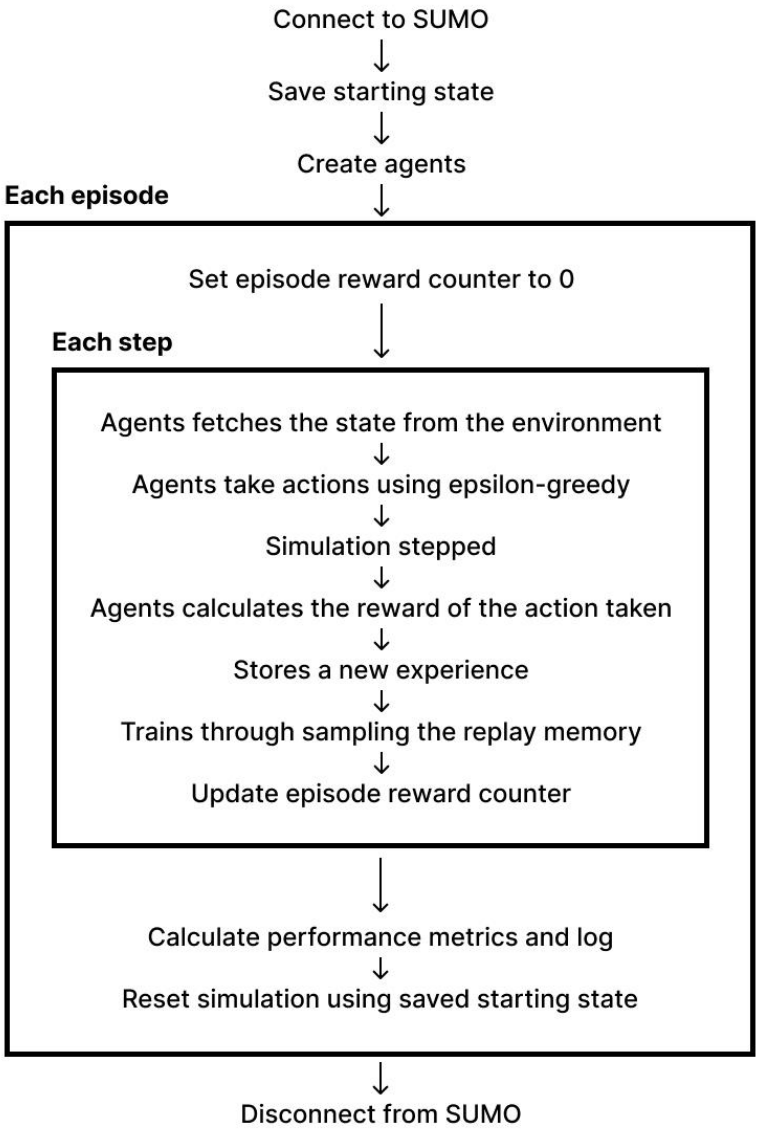
From "Figure 17", the first task performed is to start the SUMO simulation through its TraCI API and save the initial starting state of the simulation for later recall. Following, each intersection controlled by traffic lights is fetched with its corresponding identifier and related data. For each of these intersections, an instance of the agent class is instantiated which contains all methods relating to state-action, rewards, neural networks, and replay memory "Appendix F".

The artefact now enters the next phase which includes the control loop. The first layer iterates through each episode as per the initial configuration through the CLI. Within this, another loop was utilised as the simulation stepper. This contains logic that will execute in each time step of the simulation — like state-action interactions, rewards calculation, saving to replay memory, and training. Since there are also multiple agents where each contains its own set of state-actions and ANNs, another loop is needed to execute their methods for a particular time step.

Within each time step, each agent observes the current state and determines an action using the epsilon-greedy strategy. Initially, it will explore its environment through taking random actions but as the episode progresses, it tends towards exploiting through utilising stored memories and the trained ANNs. Once an action has been taken, the simulation is stepped, and the agent will now observe the new state of the SUMO environment. It will evaluate how it performed with the previous action and calculate a reward. These results form the basis for an experience which is appended to the replay memory. After which, the agent will draw a sample for the memory to train with. This marks the end of the time step loop, and the flow will return to the

loop for each episode. The final tasks performed before starting a new episode include calculating performance metrics and logging to *tensorboard* if enabled. Following these executions, the simulation will reset using the initial starting state saved at the beginning of the runtime. After all episodes have been completed, and if specified through the CLI, the ANN models for each agent will be serialised and saved in the project directory.

Figure 17 Illustrated main loop flow



Evaluating model performance

To understand the extent of the impact on traffic flow from using a DQN as the traffic light controller, key metrics were used and informed by the previous literature review. Starting with

average waiting time (AWT), this is the amount of average time the vehicle was involuntarily halted measured in seconds at the end of the episode. Measuring AWT provided insight into the severity of vehicles queuing at intersections. Likewise, average time loss (ATL) is calculated from vehicles driving at a slower speed than it was intended at the end of the episode — including AWT. This metric helped in understanding the lagging effects of poor traffic management as inefficient timings at intersections could have a cumulative effect on throughput. Vehicle count (VC) is simply the count of cars on the road network whilst episode reward (ER) is the cumulative total of rewards at all controlled intersections from every time step — essentially rating the performance of the DQN.

All these performance outcomes were measured through interfacing with the simulation using SUMO's TraCI API. As mentioned, outputs were recorded and logged at the end of each episode to *tensorboard* for storage and analysis. These results were compared with existing research to generate a discussion around the effectiveness of using DQNs in road infrastructure.

Results

All *tensorboard* data were imported into an IPython notebook (found in the submitted artefact) through the *tensorboard* API. Next, pre-processing occurred to select the relevant experiment and transform the data into their respective groups. Summary statistics were calculated for mean, quartile ranges, and standard deviation. Using the Shapiro-Wilks test, all the following results showed normal distributions; hence, an analysis of variance (ANOVA) was conducted for each performance metric. Rejecting the null hypothesis would imply that a significant difference was observed.

Average time loss

As per "Table 5", "Figure 18", and "Figure 19", time loss for MLP-2 fluctuated substantially compared to all other runs ($\sigma = 90.43$); however, it showed the lowest mean of 1355.47. Contrastingly, NO-AI showed the highest mean value of ATL being 1624.44 and having the largest maximum value of 1690.32. Performing the ANOVA revealed that the groups are significantly different to one another ($F = 15686.27, p = 3.93 \cdot 10^{-233}$).

Table 5 Summary statistics for average time loss

	Mean	SD	Min	25%	50%	75%	Max
MLP-0	1549.88	57.37	1212.71	1534.72	1553.61	1577.78	1641.71
MLP-1	1551.53	56.63	1245.06	1529.03	1552.54	1577.25	1661.41
MLP-2	1355.47	90.43	1205.69	1297.08	1326.21	1408.87	1620.80
NO-AI	1624.44	60.44	1260.24	1611.73	1636.53	1655.81	1690.32

Figure 18 Line plot of average time loss vs. episodes

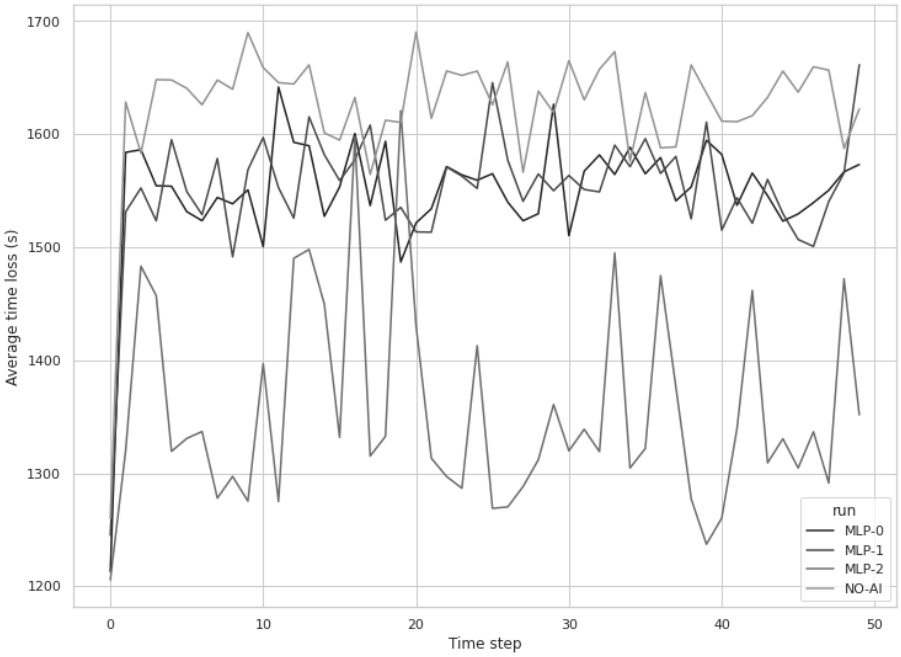
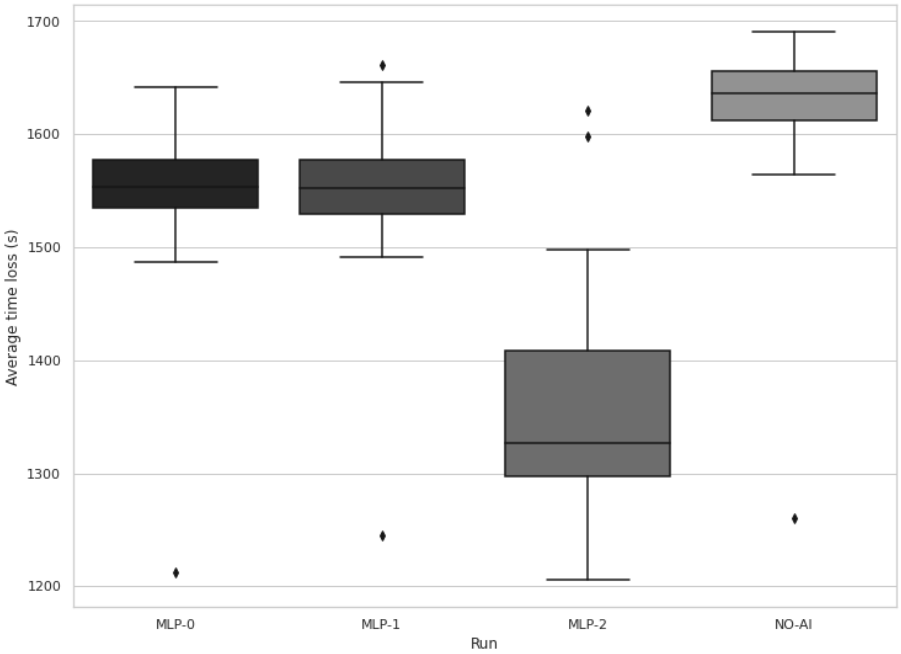


Figure 19 Box plot of average time loss



Average waiting time

From “Table 6”, “Figure 20”, and “Figure 21”, MLP-0 yielded the lowest mean across the groups but the highest deviation ($\sigma = 90.43$) with maximum and minimum values of 256.28 and 174.99 — respectively. MLP-1 and MLP-2 had higher mean values compared to MLP-0 but less than NO-AI (159.75, 170.08, and 188.14 — respectively). A similar pattern was observed with the median of each group where MLP-0 had the lowest value, followed by MLP-1, MLP-2, and NO-AI (149.95, 157.76, 169.83, and 193.33 — respectively). ANOVA revealed again a significant difference between the groups ($F = 15686.27$, $p = 3.93 \cdot 10^{-233}$).

Table 6 Summary statistics for average waiting time

	Mean	SD	Min	25%	50%	75%	Max
MLP-0	152.05	40.98	88.63	119.59	149.95	174.99	256.28
MLP-1	159.75	34.60	74.19	146.85	157.76	181.72	228.02
MLP-2	170.08	35.26	93.29	145.65	169.83	200.32	235.43
NO-AI	188.14	26.22	130.20	168.29	193.33	204.90	243.28

Figure 20 Line plot for average waiting time vs. episodes

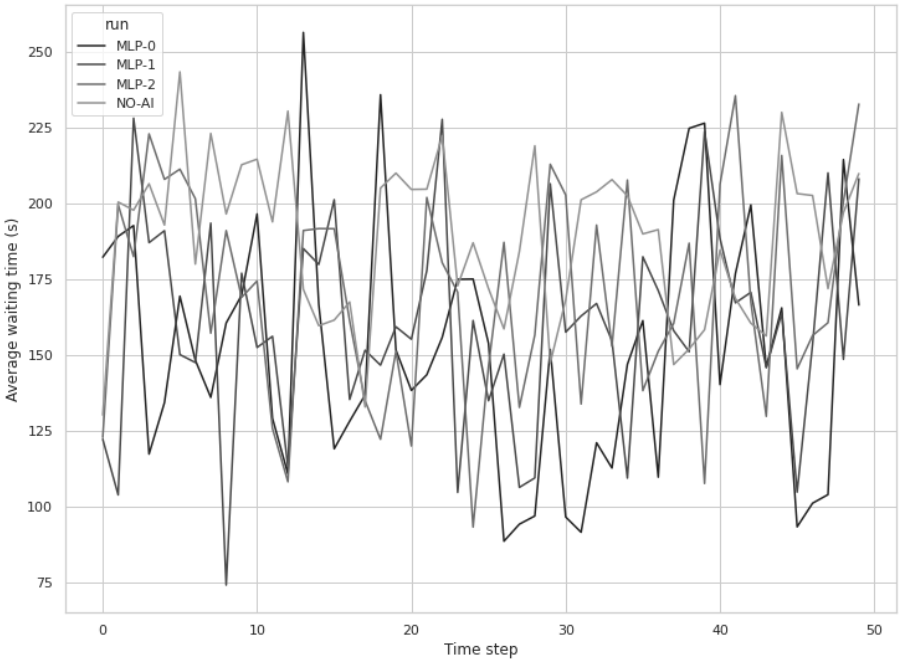
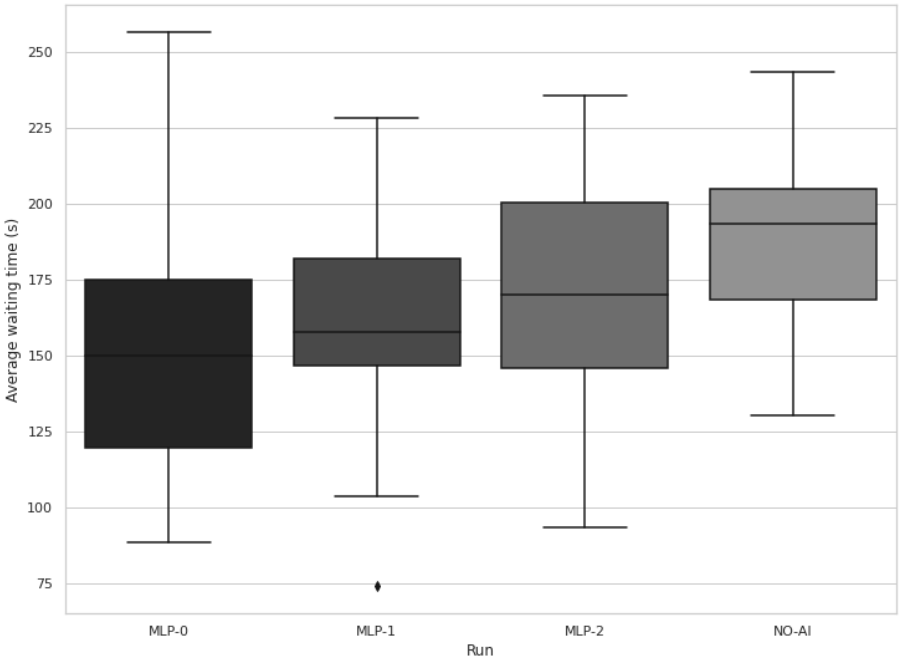


Figure 21 Box plot for average waiting time



Vehicle count

As per “Table 7”, “Figure 22”, and “Figure 23”, it was observed that MLP-2 again showed the largest deviation ($\sigma = 26.91$) but the lowest mean and minimum value (399.38 and 360.0 — respectively). Additionally, MLP-2 had a much larger interquartile range in comparison to all other groups. MLP-0 showed the second-best mean value of 440.12 followed by NO-AI and MLP-1 where both values are alike (450.06 and 450.50 — respectively). The same outcome was observed from the ANOVA where it implied the groups are significantly different from each other ($F = 15686.27, p = 3.93 \cdot 10^{-233}$).

Table 7 Summary statistics for vehicle count

	Mean	**SD**	**Min**	**25%**	**50%**	**75%**	**Max**
MLP-0	440.12	12.92	379.0	434.0	442.0	447.00	463.0
MLP-1	450.50	13.01	383.0	447.0	451.5	458.00	474.0
MLP-2	399.38	26.91	360.0	383.0	389.0	414.25	466.0
NO-AI	450.06	12.12	389.0	446.0	451.0	456.75	469.0

Figure 22 Line plot of vehicle count vs. episodes

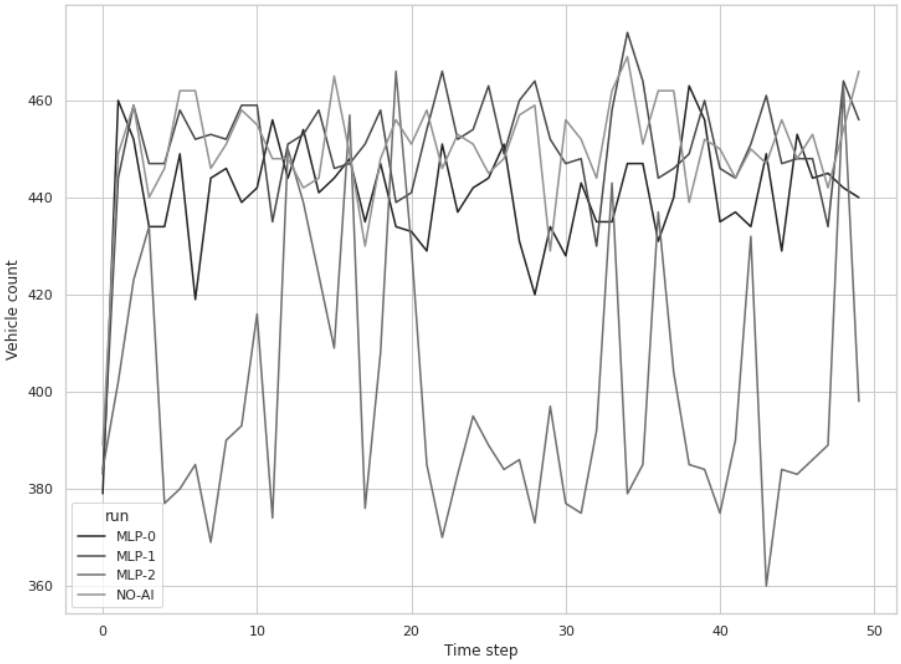
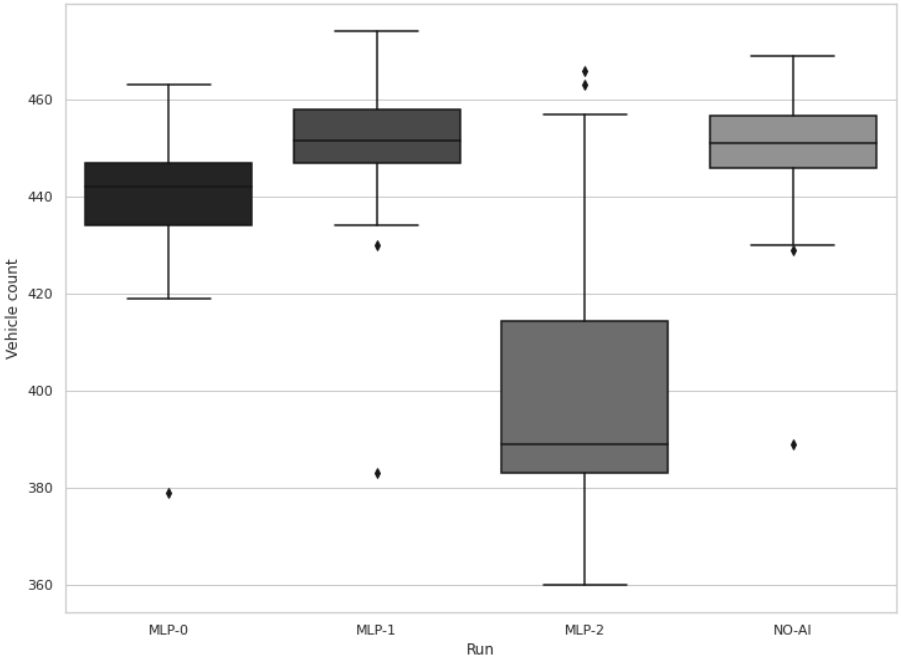


Figure 23 Box plot of vehicle count



Episode rewards

Excluding NO-AI as rewards are not relevant to the FT approach, it was observed that MLP-2 demonstrated the highest mean value compared to MLP-0 and MLP-1 (4084.75, 2044.09, and 2926.75 — respectively). Again, MLP-2 displayed the largest deviation ($\sigma = 694.66$) and

interquartile range as per “Table 8”, “Figure 24”, and “Figure 25”. The results from the ANOVA showed significant differences were observed between the groups ($F = 9411.76$, $p = 7.23 \cdot 10^{-156}$).

Table 8 Summary statistics for episode rewards

	Mean	**SD**	**Min**	**25%**	**50%**	**75%**	**Max**
MLP-0	2044.09	507.01	1485.36	1886.41	1991.33	2089.21	5367.29
MLP-1	2926.75	351.50	1835.32	2763.24	2933.40	3090.48	4300.40
MLP-2	4084.75	694.66	2269.03	3712.61	4366.30	4574.43	5134.87

Figure 24 Line plot of episode rewards vs. episodes for all DQN models only

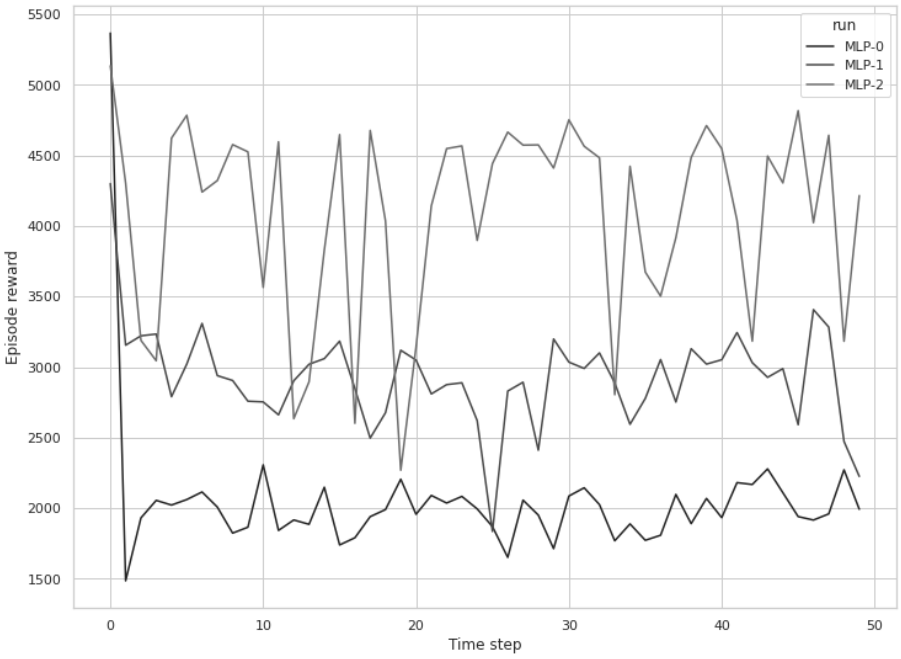
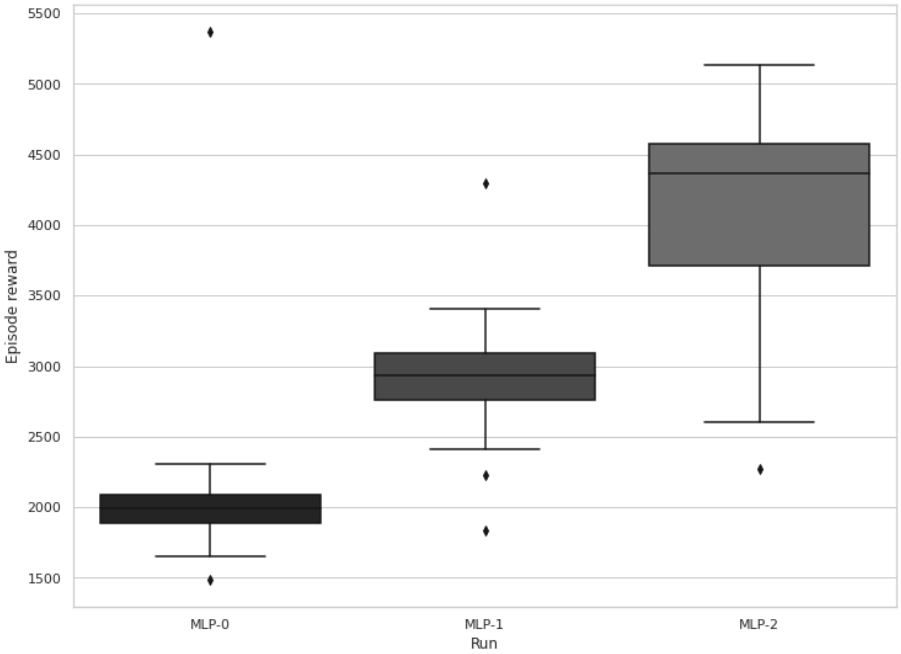


Figure 25 Box plot of episode rewards for all DQN models only



Comparison to baseline

Looking at the performance metrics, improvements were observed from using a DQN approach compared to an FT method used in traditional traffic light control systems. From "Table 9", MLP-2 reduced ATL by 16.56% compared to MLP-0 and MLP-1 where both had similar outcomes (-4.59% and -4.49% — respectively). Contrastingly, MLP-2 had the worst reduction in AWT whilst MLP-0 had the highest (-9.60% and -19.18% — respectively). Nonetheless, MLP-2 again showed the best outcome for VC by reducing the number of vehicles at the end of the simulation by 11.26% compared to the NO-AI approach. Interestingly, MLP-1 performed marginally worse than the NO-AI scenario with increasing VC by 0.10%. Finally, regarding the performance of the DQN models, MLP-2 showed the largest ER with a mean value of 4084.75 across all episodes. MLP-1 followed with 2926.75 and MLP-0 with 2044.09.

Table 9 Performance metric averages (with % change from using NO-AI)

	Avg. time loss (s)	**Avg. waiting time (s)**	**Vehicle count**	**Episode reward**
MLP-0	1549.88 (-4.59%)	152.05 (-19.18%)	440.12 (-2.21%)	2044.09
MLP-1	1551.53 (-4.49%)	159.75 (-15.09%)	450.50 (0.10%)	2926.75
MLP-2	1355.47 (-16.56%)	170.08 (-9.60%)	399.38 (-11.26%)	4084.75
NO-AI	1624.44	188.14	450.06	-

Discussion

Summarising the key rationales of this work, the goal is to extend the existing pool of research by considering additional constraints. Therefore, the research question/statement is to understand the performance of a multi-intersection RL model that utilises the vehicle detection range of intersections in a real-world network. Other researchers and authorities could use this research in areas like edge-based computing where models are deployed locally at the data source — the signalled intersections. Again, the hypothesis was that the implementation that this project has taken will observe an improvement in all performance metrics compared to an FT approach. However, it was anticipated that the outcomes would not be as substantial as those from previous authors due to the additional constraints of lane area detectors and the real-world environment.

From the results, it seems that the hypothesis holds for the expectation that improvements were observed in comparison to the FT method but less in comparison to previous work. As mentioned, MLP-2 showed the greatest improvements in ATL and VC (16.56 and 11.26 — respectively) where it also yielded the largest sum of ER at 4084.75. Previous research was able to produce a larger reduction in waiting time of 23% compared to the best improvement observed in this project which was at 19.18% (Choe *et al.* 2018). Differences between this research and the aforementioned article were the inclusion of an LSTM in the neural network used for the DQN and differences in the structure of the environment's state. This could suggest that traffic light control systems using DQNs should consider a model that can retain information for a longer duration like an LSTM. Nonetheless, it is important to consider the trade-off between using a more complex model that could increase the training time — subsequently the hardware and costs. It can be argued that achieving a slightly better outcome in waiting time whilst requiring more hyperparameter tuning and the above disadvantages may not be suitable.

Another comparable study reported even higher improvements in waiting time. The authors reported a 48% improvement whilst comparing it to an FT method (Joo and Lim 2021). Interestingly, unlike this study, it was also reported that the standard deviation was 45% lower than the FT group. In comparison, the deviation with the best model for this performance metric in this study reported an increase of 43.93%. Though the authors did not specify the exact structure of the neural network used, it is plausible that these differences in outcomes could

relate to the realism of the simulation. In Joo and Lim (2021), a singular intersection of 8 actions was implemented. Similarly, the authors opted to use a vector of lane queue lengths as the state input for the DQN. Asides from that, the methodologies greatly differed which allows for new discussion around the performance differences between a study that uses a representation of real-world infrastructure and one that does not. In this case, using an artificially created singular intersection with more choice of actions yielded much better outcomes.

As introduced in the methodology, this study has implemented a dynamic approach to DQN state-action initialisation — meaning that each intersection is its own agent with a particular set of states and actions. This was an attempt to introduce a new approach to isolate each intersection so that they did not depend on the states of surrounding intersections. Having a model train independently gains an advantage in a real-world scenario where traffic light systems could go offline — and would not affect this agent. A model that was trained with states from surrounding agents may result in a poor choice of actions when traffic lights are not working. However, some authors have taken this opposing method as using surrounding data can also improve the performance of smart intersections across the network. It was discovered that an extended DQN solution which included a database that distributed data amongst all agents reported at most a 16.7% decrease in waiting time compared to a simpler DQN approach (Liu *et al.* 2017). Nonetheless, this could gather scalability issues as the size of the road network increases — meaning more considerations would be needed to implement such a method in the real world. For instance, the storing and distribution of the intersection data across the network, load balancing during high/low traffic hours, and preventing the DQN from generalising too much due to a large amount of data. In contrast, using a localised method could help avoid these issues — by training a local DQN that can recognise traffic patterns and optimise signal timings at any given moment.

Other studies have also reported positive outcomes for AWT and ATL from using a more complex DQN. For instance, using a method the authors called a ‘risk-sensitive’ approach gathered an 18.2% and 13.2% reduction in AWT and ATL — respectively (R *et al.* 2022). The authors introduced the concept of distributional reinforcement learning (DRL) which differs from RL on the basis that the algorithm returns the distribution rather than a scalar value at the given state. The distribution returned by the DRL represents the probability of different return values that the agent can obtain (Bellemare *et al.* 2022). In short, the model differed

substantially from the simpler MLPs used in this research. Hence, it could be plausible to assume that their improvements were greater due to the inclusion of a more complex network — much like the previous discussion about using LSTMs. However, from the approach that they have specified, it could be a combination of the DRL and the much higher detector range of 70m used in comparison to the 15–20m range used in this study. Yet again, this supports the hypothesis as it seems that adding more constraints to the visibility of the DQN reduced its effectiveness. Furthermore, R et al. (2022) used fixed durations for each phase of the traffic light where the DQN would decide which phase to run sequentially. Controlling this duration could have resulted in a more stable and consistent flow of traffic — rather than letting the agent decide when and for how long each phase will last. Nonetheless, the drawbacks of using a partial timing method could be that vehicles are halted at traffic lights for longer than needed.

A common pattern across previous research revealed that complex networks generally obtained better performance outcomes. This mostly held true in this study's experiment results. As per "Figure 25", for each increasingly more complex variant of the MLP used in the DQN, higher rewards were recorded where MLP-2 showed the largest value of ER. With each hidden layer in MLP-2, the number of neurons reduced until the output layer. This is to generalise the information passing through the network where the weights were adjusted accordingly to the importance of the feature and its derivations. Though the neural networks used in previous studies varied — this pattern was common and was used to extend CNNs and LSTMs (Du *et al.* 2019, Ge *et al.* 2019, Zhao *et al.* 2019). Once again, the rationale for using an MLP in this study was due to the simplistic nature of the state input as only lane queue lengths were considered. Hence, implementing a CNN or LSTM seemed computationally expensive for this particular usage.

Conclusion

In short, the outcome was as expected from the hypothesis that improvements would be observed compared to the baseline but not in comparison to previous research. From the findings, it seems that using a deep Q-learning agent built with neural networks that has more hidden layers which generalise passing information can produce good improvements. The solution implemented in this study was able to reduce average time loss from driving at slower speeds due to traffic by 16.56%. The average waiting time of vehicles halted at an intersection was also reduced by 19.18%, and the number of vehicles on the road network was less by 11.26%.

Limitations

As previously eluded, some limitations exist in this study's methodology and implementation. For instance, there are no yellow phases between each active phase for the traffic light intersections. This was to reduce the complexity of the simulation and subsequently impacted the realism of the scenario. Moreover, other vehicle types and pedestrians were not considered. Also, the state gathered from the environment could be extended with more information. Doing this adds more features for the model to analyse — distinguishing relationships between them. Adding these would generalise the model but broaden the scope of this study excessively. Since the implementation of the agents in this project differed from other studies due to dynamically initialised neural networks, a more controlled approach was taken to assess the suitability of this implementation.

Future research

Some key areas to tackle include the ones mentioned in the limitations. As of the current literature, none found in the review depicted the use of yellow phases in multi-intersection DQNs where state and action were dynamically instantiated. A possible solution could include using a queue to execute the phases as the agent predicts the action to take with yellow phases inserted between each action. However, this could create a situation where the agent appends a new phase to this queue at every time step which is not desired as it could create a scenario where queued phases are executed much later than intended. Alternatively, an implementation could include a simple guard logic that returns a yellow phase action whenever the agent switches active phases. Moreover, other vehicle types and pedestrians could also be considered in upcoming research. These would improve the implementation shown in this project — adding to the DQN's ability to generalise the understanding of its states and subsequently becoming more adaptive to real-world scenarios.

Project Management

(adapted from previous work, Lau 2022)

Clear planning has assisted in meeting the requirements for this project within the given timeframe following the project timeline as per "Figure 26". The pilot study justified the need for this research and its feasibility. This report began with the construction of the training environment using SUMO and its first-party tools utilising OSM to generate the road network.

The goal of this phase was to determine a suitable environment that captures the key aspects of road networks that causes congestion at intersections. Subsequently, the creation of the agent followed. Using a combination of standard and third-party libraries, the software artefact facilitated the training of the agent, simulation stepping, replay memory sampling, gathering observations from the SUMO environment, and taking action using the predicted Q-values.

A consistent workflow was present throughout the project due to the implementation of DevOps tools like pre-commits, linters, and type-checkers. The source code can be accessed through GitHub as a remote repository. Additionally, a Dockerfile is also included which can help run the project in a pre-specified environment and minimise the steps needed to setup. Installing the dependencies and following the instructions will obtain the same result. Through trial-and-error, debugging, and mock experiments, the training and optimisation of the model began just after halfway through the project — where results were logged using *tensorboard*. Hyperparameters were tuned following and inspired by specifications from previous research.

After gathering the results, statistical analysis was performed between each type of ANN model used for the agent and evaluated its performance in comparison to the control — being the no AI implementation. The write-up began starting with the order of introduction, methods, discussion, conclusion, and finishing the abstract last. The final step was to add in any supplementary materials like appendices and proof the draft copies — which was conducted in the final days before submission.

“Table 10” illustrates the workflow of this project, its expectation and the actual outcome for each task. It is clear that from the beginning, tasks were completed on-time — sometimes earlier. However, as the project progressed into its later stages, issues started arising. For instance, the training and optimising of the model took longer than expected due to bugs and poor hyperparameter tuning. Personal time constraints arose from this period which made meeting the expected end date difficult. Nonetheless, the training of the DQN was subsequently resolved by running all experiments on the local machine. Initially, a cloud service named *Hetzner* was used to train the model on a billed dedicated cluster. However, the hardware was not sufficient and pricing for more resources was not acceptable. Hence, some refactoring was required which took more time due to SUMO requiring a specific setup depending on the system. In short, much of the delays were caused by an oversight in regard to the hardware requirements needed for training and also some unexpected problems with the artefact. To

prevent a similar scenario in the future, unit testing could help identify areas that do not conform to expectations — whilst a coverage report can help understand which areas of the source code could be vulnerable to package upgrades.

Figure 26 Gantt of the project timeline

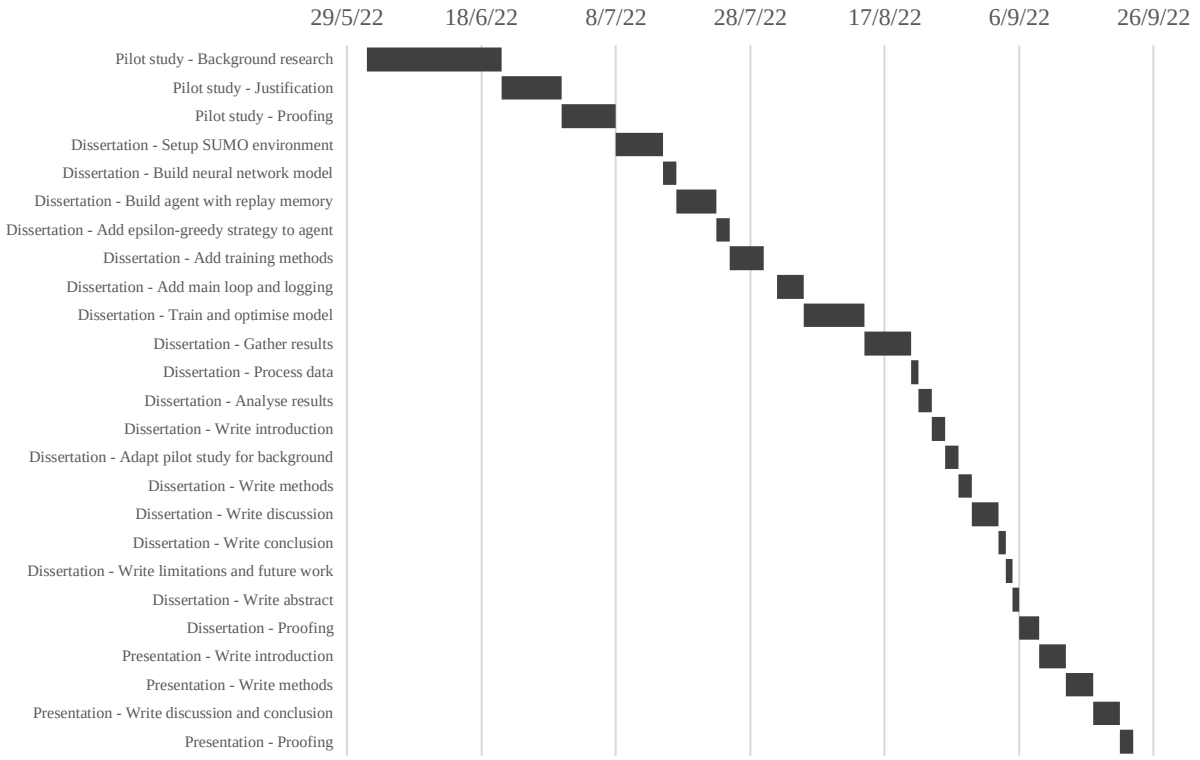


Table 10 List of tasks, their expected hours, and the actual time spent on the report

Task	**Expected Hours**	**Actual Hours**
Setup SUMO environment	15	10
Build neural network model	10	4
Build agent with replay memory	20	4
Add epsilon-greedy strategy to agent	5	2
Add training methods	5	4
Add main loop and logging	5	6
Train and optimise model	20	27
Gather results	20	4
Process data	6	15
Analyse results	10	4
Write introduction	10	12
Adapt pilot study for background	10	4
Write methods	10	23
Write discussion	20	21
Write conclusion	2	1
Write limitations and future work	2	1
Write abstract	2	1
Proofing	5	10

References

BELLEMARE, M., W. DABNEY and M. ROWLAND, 2022. *Distributional Reinforcement Learning*. MIT Press

CHEN, Y. and E. KULLA, 2019. A Deep Q-Network with Experience Optimization (DQN-EO) for Atari's Space Invaders. In: L. BAROLLI et al., eds. *Web, Artificial Intelligence and Network Applications*. Cham: Springer International Publishing, pp. 351–361

CHOE, C.-J. *et al.*, 2018. Deep Q Learning with LSTM for Traffic Light Control. In: *2018 24th Asia-Pacific Conference on Communications (APCC)*. pp. 331–336

DABNEY, W., G. OSTROVSKI and A. BARRETO, 2020. Temporally-Extended ε-Greedy Exploration

DEPARTMENT FOR TRANSPORT, 2021. *Road traffic statistics - Summary statistics* [viewed 24 June 2022]. Available from: https://roadtraffic.dft.gov.uk/summary

DU, Y. *et al.*, 2019. RA-TSC: Learning Adaptive Traffic Signal Control Strategy via Deep Reinforcement Learning. In: *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. pp. 3275–3280

FAN, J. *et al.*, 2020. A Theoretical Analysis of Deep Q-Learning. In: *Proceedings of the 2nd Conference on Learning for Dynamics and Control*. PMLR, pp. 486–489

FRANÇOIS-LAVET, V., R. FONTENEAU and D. ERNST, 2016. How to Discount Deep Reinforcement Learning: Towards New Dynamic Strategies

GARG, D., M. CHLI and G. VOGIATZIS, 2018. Deep Reinforcement Learning for Autonomous Traffic Light Control. In: *2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE)*. pp. 214–218

GE, H. *et al.*, 2019. Cooperative Deep Q-Learning With Q-Value Transfer for Multi-Intersection Signal Control. *IEEE Access*, 7, 40797–40809

GE, Z., 2020. Reinforcement Learning-based Signal Control Strategies to Improve Travel Efficiency at Urban Intersection. In: *2020 International Conference on Urban Engineering and Management Science (ICUEMS)*. pp. 347–351

GURESEN, E. and G. KAYAKUTLU, 2011. Definition of artificial neural networks with comparison to other networks. *Procedia Computer Science*, 3, 426–433

HAN, D., Q. LIU and W. FAN, 2018. A new image classification method using CNN transfer learning and web data augmentation. *Expert Systems with Applications*, 95, 43–56

HARA, K., D. SAITO and H. SHOUNO, 2015. Analysis of function of rectified linear unit used in deep learning. In: *2015 International Joint Conference on Neural Networks (IJCNN)*. pp. 1–8

HE, L. *et al.*, 2021. A Spatial-Temporal Graph Attention Network for Multi-intersection Traffic Light Control. In: *2021 International Joint Conference on Neural Networks (IJCNN)*. pp. 1–8

JOO, H. and Y. LIM, 2021. Intelligent Traffic Signal Control System using Deep Q-network. In: *2021 IEEE 3rd Eurasia Conference on IOT, Communication and Engineering (ECICE)*. pp. 285–287

KINGMA, D.P. and J. BA, 2017. Adam: A Method for Stochastic Optimization
LIU, Y., L. LIU and W.-P. CHEN, 2017. Intelligent traffic light control using distributed multi-agent Q learning. In: *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*. pp. 1–8

OPENSTREETMAP, 2022. *OpenStreetMap* [viewed 6 July 2022]. Available from: https://www.openstreetmap.org/about

O'SHEA, K. and R. NASH, 2015. An Introduction to Convolutional Neural Networks
PANG, H. and W. GAO, 2019. Deep Deterministic Policy Gradient for Traffic Signal Control of Single Intersection. In: *2019 Chinese Control And Decision Conference (CCDC)*. pp. 5861–5866

R, A., M. KRISHNAN and A. KEKUDA, 2022. Intelligent Traffic Control System using Deep Reinforcement Learning. In: *2022 International Conference on Innovative Trends in Information Technology (ICITIIT)*. pp. 1–8

ROSENBLATT, F., 1958. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6), 386–408

ROSYADI, A.R., T.A.B. WIRAYUDA and S. AL-FARABY, 2016. Intelligent traffic light control using collaborative Q-Learning algorithms. In: *2016 4th International Conference on Information and Communication Technology (ICoICT)*. pp. 1–6

SHINDE, M. *et al.*, 2021. Multiple Intersection Traffic Control using Reinforcement Learning. In: *2021 2nd Global Conference for Advancement in Technology (GCAT)*. pp. 1–4

SUGIYAMA, M., 2015. *Statistical Reinforcement Learning: Modern Machine Learning Approaches*. CRC Press

SUMO, 2022. *Safety - SUMO Documentation* [viewed 4 July 2022]. Available from: https://sumo.dlr.de/docs/Simulation/Safety.html

TAN, T., T. CHU and J. WANG, 2020. Multi-Agent Bootstrapped Deep Q-Network for Large-Scale Traffic Signal Control. In: *2020 IEEE Conference on Control Technology and Applications (CCTA)*. pp. 358–365

WANG, Y., 2017. A new concept using LSTM Neural Networks for dynamic system identification. In: *2017 American Control Conference (ACC)*. pp. 5324–5329

WANG, Y. *et al.*, 2022. STMARL: A Spatio-Temporal Multi-Agent Reinforcement Learning Approach for Cooperative Traffic Light Control. *IEEE Transactions on Mobile Computing*, 21(6), 2228–2242

WELLS, L. and T. BEDNARZ, 2021. Explainable AI and Reinforcement Learning—A Systematic Review of Current Approaches and Trends. *Frontiers in Artificial Intelligence*, 4

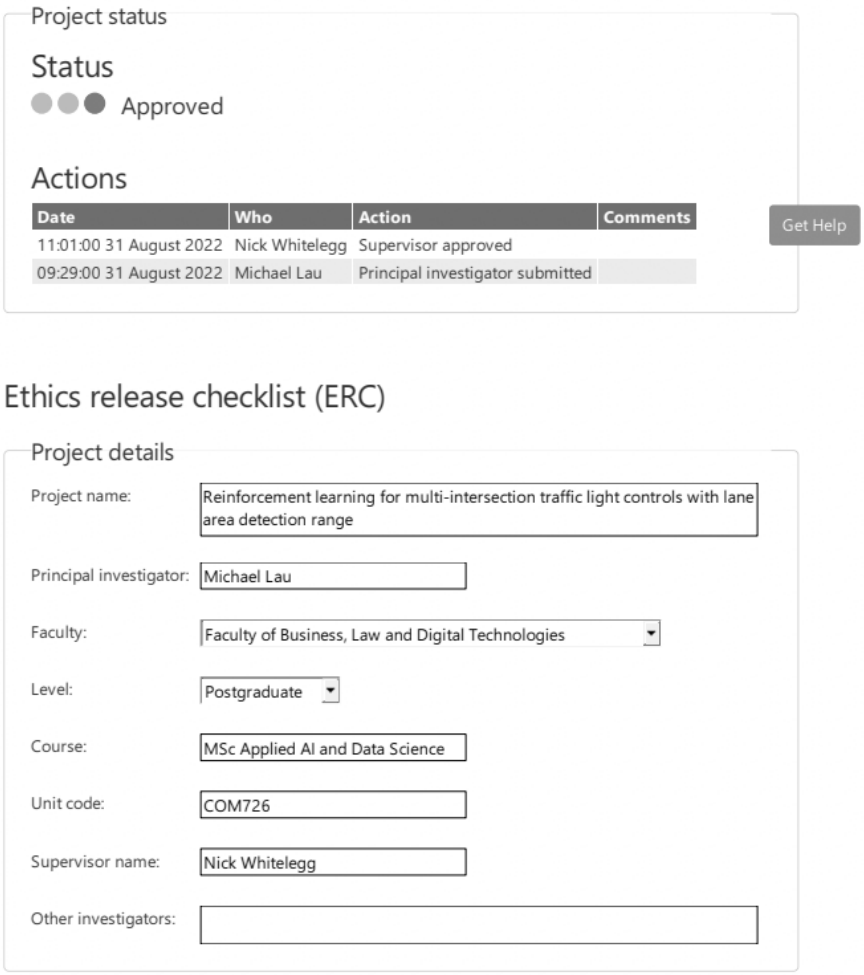
WU, T. *et al.*, 2020. Road Intersection Model Based Reward Function Design in Deep Q-Learning Network for Traffic Light Control. In: *2020 5th International Conference on Robotics and Automation Engineering (ICRAE)*. pp. 182–186

WU, T. *et al.*, 2020. Multi-Agent Deep Reinforcement Learning for Urban Traffic Light Control in Vehicular Networks. *IEEE Transactions on Vehicular Technology*, 69(8), 8243–8256

ZHAO, T., P. WANG and S. LI, 2019. Traffic Signal Control with Deep Reinforcement Learning. In: *2019 International Conference on Intelligent Computing, Automation and Systems (ICICAS)*. pp. 763–767

Appendices

Appendix A: Ethical approval



Project status

Status

Approved

Actions

Date	Who	Action	Comments
11:01:00 31 August 2022	Nick Whitelegg	Supervisor approved	
09:29:00 31 August 2022	Michael Lau	Principal investigator submitted	

Get Help

Ethics release checklist (ERC)

Project details

Project name: Reinforcement learning for multi-intersection traffic light controls with lane area detection range

Principal investigator: Michael Lau

Faculty: Faculty of Business, Law and Digital Technologies

Level: Postgraduate

Course: MSc Applied AI and Data Science

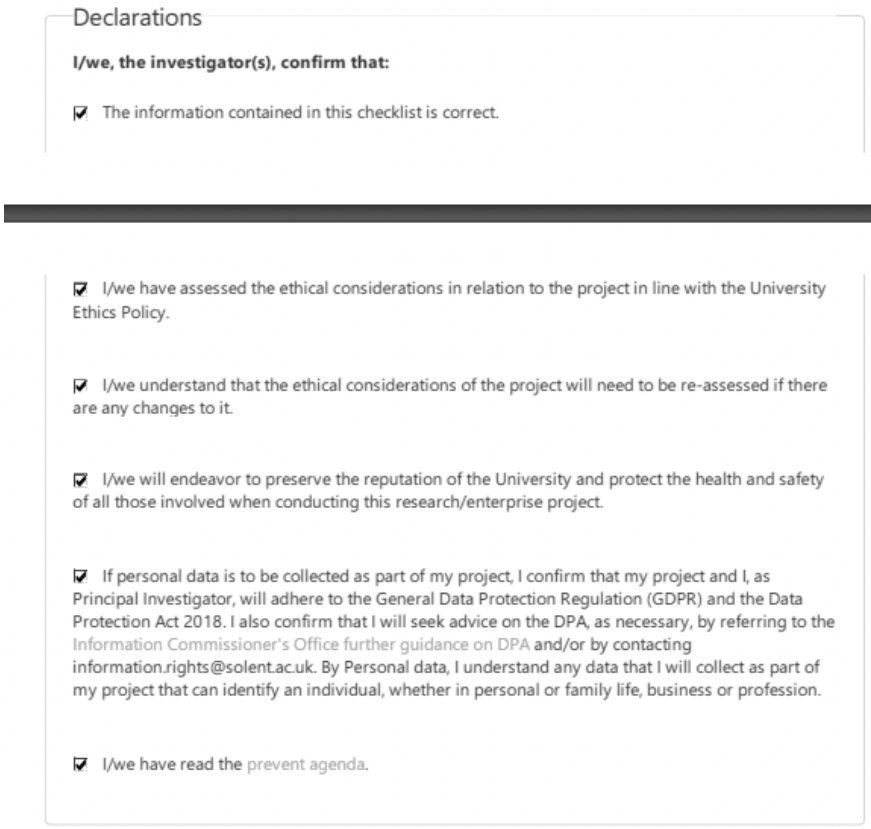
Unit code: COM726

Supervisor name: Nick Whitelegg

Other investigators:

Checklist

Question	Yes	No
Q1. Will the project involve human participants other than the investigator(s)?	○	◉
Q1a. Will the project involve **vulnerable participants** such as children, young people, disabled people, the elderly, people with declared mental health issues, prisoners, people in health or social care settings, addicts, or those with learning difficulties or cognitive impairment either contacted directly or via a **gatekeeper** (for example a professional who runs an organisation through which participants are accessed; a service provider; a care-giver; a relative or a guardian)?	○	◉
Q1b. Will the project involve the use of **control groups** or the **use of deception**?	○	◉
Q1c. Will the project involve any **risk to the participants' health** (e.g. intrusive intervention such as the administration of drugs or other substances, or vigorous physical exercise), or involve psychological stress, anxiety, humiliation, physical pain or discomfort to the investigator(s) and/or the participants?	○	◉
Q1d. Will the project involve **financial inducement** offered to participants other than reasonable expenses and compensation for time?	○	◉
Q1e. Will the project be carried out by individuals unconnected with the University but who wish to use staff and/or students of the University as participants?	○	◉
Q2. Will the project involve sensitive materials or topics that might be considered offensive, distressing, politically or socially sensitive, deeply personal or in breach of the law (for example criminal activities, sexual behaviour, ethnic status, personal appearance, experience of violence, addiction, religion, or financial circumstances)?	○	◉
Q3. Will the project have detrimental impact on the environment, habitat or species?	○	◉
Q4. Will the project involve living animal subjects?	○	◉
Q5. Will the project involve the development for export of 'controlled' goods regulated by the Export Control Organisation (ECO)? (This specifically means military goods, so called dual-use goods (which are civilian goods but with a potential military use or application), products used for torture and repression, radioactive sources.) Further information from the Export Control Organisation	○	◉
Q6. Does your research involve: the storage of records on a computer, electronic transmissions, or visits to websites, which are associated with terrorist or extreme groups or other security sensitive material? Further information from the Information Commissioners Office	○	◉



Declarations

I/we, the investigator(s), confirm that:

☑ The information contained in this checklist is correct.

☑ I/we have assessed the ethical considerations in relation to the project in line with the University Ethics Policy.

☑ I/we understand that the ethical considerations of the project will need to be re-assessed if there are any changes to it.

☑ I/we will endeavor to preserve the reputation of the University and protect the health and safety of all those involved when conducting this research/enterprise project.

☑ If personal data is to be collected as part of my project, I confirm that my project and I, as Principal Investigator, will adhere to the General Data Protection Regulation (GDPR) and the Data Protection Act 2018. I also confirm that I will seek advice on the DPA, as necessary, by referring to the Information Commissioner's Office further guidance on DPA and/or by contacting information.rights@solent.ac.uk. By Personal data, I understand any data that I will collect as part of my project that can identify an individual, whether in personal or family life, business or profession.

☑ I/we have read the prevent agenda.

Appendix B: Replay memory implementation

```
class ReplayMemory:
    """Stores past experiences and handles adding and sampling from memory."""

    def __init__(self, replay_size: int) -> None:

        self.memory: deque[Experience] = deque([], maxlen=replay_size)

    def __len__(self) -> int:
        """Returns the length of the replay memory."""

        return self.memory.__len__()

    def push(self, experience: Experience) -> None:
        """Pushes a new experience into the memory.

        Args:
            experience (Experience): New experience to add.
        """

        self.memory.append(experience)

    def sample(self, batch_size: int) -> list[Experience]:
        """Gets a sample of existing memory with specified size.

        Args:
            batch_size (int): The number of experiences to draw.

        Returns:
            list[Experience]: The list of experiences drawn.
        """

        return sample(self.memory, batch_size)
```

Appendix C: Experience named tuple

```
Experience = namedtuple("Experience", ("state", "action", "next_state", "reward"))
```

Appendix D: DQN target network training implementation

```
# List of all previous experiences
exps = Experience(*zip(*self.memory.sample(self.batch_size)))

state_exps = torch.cat(exps.state)
action_exps = torch.cat(exps.action)
reward_exps = torch.cat(exps.reward)
next_states = torch.cat(exps.next_state)

# Q-values from current state-action
sa_values = self.net(state_exps).gather(1, action_exps)

# Predicted next state q-values
next_state_values = self.target_net(next_states).max(1)[0].detach()
expected_sa_values = (next_state_values * self.gamma) + reward_exps

# Loss function
criterion = MSELoss()
loss = criterion(sa_values, expected_sa_values.unsqueeze(1))
```

Appendix E: A custom command-line interface (CLI) for the artefact

```
CLI for SUMO RL

options:
  -h, --help          show this help message and exit
  --episodes       The number of episodes to run.
  --max-step       The maximum number of steps in each episode.
  --gui              Sets SUMO to launch with GUI.
  --no-ai            Sets SUMO to not use the DQN.
  --epsilon-max    Starting/maximum value of epsilon.
  --epsilon-min    Minimum value of epsilon.
  --epsilon-decay  The proportional decay of epsilon in steps.
  --gamma          The discount rate.
  --batch-size     The number of replay memory experiences to draw from.
  --replay-size    The size of the replay memory.
  --sync-rate      The frequency of syncing the target net in steps.
  --log              Logs each episode to Tensorboard.
  --save-models      Saves the DQN model.
  --model-variant  The variant of the model.
```

Appendix F: Instantiate a list of agents for every traffic light intersection

```
# All TLS agents
TLS_AGENTS: tuple[Agent, ...] = tuple(
    Agent(
        TrafficLightSystem(
            tls_id,
            traci.trafficlight.getControlledLanes(tls_id),
            traci.trafficlight.getAllProgramLogics(tls_id)[0].phases,
        ),
        agent_config,
    )
    for tls_id in traci.trafficlight.getIDList()
)
```

Appendix G: Main loop for reinforcement learning agents

```
# Main simulation loop
for ep in range(args.episodes):
    eps_reward: float = 0

    # Episode simulation stepper
    for step in range(args.max_step):

        if args.no_ai:
            traci.simulationStep()
            continue

        # Prepares action for each TLS agent
        sa_pairs = [agent.prepare_step(step) for agent in TLS_AGENTS]

        traci.simulationStep()

        # In the next step, evaluate the action taken
        for idx, agent in enumerate(TLS_AGENTS):
            state, action = sa_pairs[idx]
            next_state, reward = agent.evaluate_step(state)

            # Saves the experience
            agent.memory.push(Experience(state, action, next_state, reward))

            # Performs training with epsilon greedy
            agent.train(step)

            # Updates this episode's reward
            eps_reward += reward.reshape(-1)[0].item()
```