UNIVERSITY OF HULL

Exploring Deep Q-Networks in Autonomous Highway Navigation with the CARLA Simulator

Final report

Submitted for the BSc/MEng in

Computer Science Integrated Masters

April 2025

by

Archie Hull

Word count: 12,034

Abstract

This project explores the application of Deep Q-Networks to autonomous highway driving using the CARLA simulator. The goal was to train a reinforcement learning agent to navigate a multi-lane highway environment safely and efficiently using image-based input and discrete control actions. The project involved the design and implementation of a training pipeline incorporating experience replay, a target network and a modular reward structure. Several convolutional neural network architectures were tested, with the 64x3 model demonstrating the most stable learning behaviour under CPU constraints.

Due to technical limitations created by compatibility issues, the system failed to achieve real-time responsiveness or consistent autonomous behaviour. However, the infrastructure and components were successfully implemented, and foundational progress was made in integrating reinforcement learning with a high-fidelity simulation environment. The report critically evaluates the model performance and contributes a reproducible framework for reinforcement learning research in CARLA under constrained conditions, offering insights into both the potential and challenges of training autonomous agents in simulated environments.

Contents

1 Introduction

1.1 Background to the project

The application of reinforcement learning in autonomous vehicle systems has been a growing area of interest within the automotive industry, given the increasing demand for intelligent transport solutions capable of adapting to dynamic and complex environments. In contrast to more traditional rule-based or supervised learning models, reinforcement learning offers the potential to develop systems capable of making decisions in complex real-world traffic scenarios.

The CARLA simulator is used to explore the implementation of reinforcement learning models, with the aim of improving vehicle performance in traffic environments. The project seeks to simulate a highway driving environment, where an autonomous vehicle agent learns to navigate at the highest possible speed, while adhering to traffic rules and avoiding collisions.

Inspiration is drawn from similar simulation-based reinforcement learning implementations in this project, most notably the Massachusetts Institute of Technology's DeepTraffic competition, which employed Q-learning variants to control the navigation of an autonomous agent through a simplified grid-based highway model. While DeepTraffic provided a valuable foundation for understanding the efficacy of discrete-state reinforcement learning, CARLA offers a significantly more realistic and extensible environment, including support for perception, sensor noise, continuous action space, complex traffic is and interaction with a Python client. These attributes make CARLA more appropriate for simulating real-world constraints and testing the scalability of autonomous vehicle algorithms.

This project is positioned to contribute to the research conducted by the broader autonomous vehicle industry and their goal of reducing human error and improving road safety through the use of artificial intelligence. As real-world testing poses significant safety, ethical and financial risks, simulation-based experimentation offers a safer and more cost-effective avenue for developing, testing and validating autonomous vehicle policies.

1.2 Research question

The primary research question for this project is: *How can deep reinforcement learning algorithms be used to improve the performance of autonomous vehicles in a highway driving environment?* This is an investigation that will be conducted through the research, implementation and analysis of reinforcement learning models, evaluating their capacity to produce safe and efficient driving practices within simulated highway environments. Secondary considerations include the impact of Deep Q-Network configurations on training stability and performance, and the ethical, safety and practical implications of deploying learned policies in real-world environments.

The models used in this project will be designed to balance safety and efficiency but will not focus on the issues more closely associated with real-world environments, such as pedestrian avoidance, adaptivity to weather conditions and the impact of driver and passenger behaviour.

1.3 Aims and objectives

The primary aim of this project is to investigate the use of deep reinforcement learning techniques within the CARLA simulator and the effectiveness of Deep Q-Networks for autonomous vehicles in a highway scenario.

Core objectives include:

- Establishing a simulated highway environment in CARLA that can reflect multi-lane traffic behaviour.
- Implementing a baseline Deep Q-Network agent capable of making real-time decisions regarding speed and positioning.
- Evaluating performance using measurable metrics, such as average speed, model loss, average reward, episode time, collision rate.
- Conducting comparative experiments with different architectures to assess their impact on learning capability and model performance.
- Assessing the capability for models to perform across varied traffic environments and the implications of real-world deployment.

A successful model will be able to avoid collisions and sustain a safe and high average velocity while navigating a traffic environment. The conclusion of this project should provide an informed evaluation of the suitability of Deep Q-Network based models for autonomous highway navigation, alongside insights into the model's limitations, areas of improvement and recommendations for future research directions.

In addition to these technical goals, this project also brings academic contribution to the artificial intelligence and automation industry. By adapting a Deep Q-Network to a complex, dynamic simulation such as CARLA, the application of abstract reinforcement learning research and applied autonomous vehicle control can be explored. This contributes to the understanding of how discrete-action reinforcement agents perform in high-speed, multi-vehicle driving scenarios. The engagement in design for reward function, state and exploration strategy, offers insights into the sensitivity of Deep Q-Networks to these components in safety-critical domains. By evaluating generalisation across varied traffic densities and environmental conditions, the work provides a foundation for assessing robustness in learning-based driving agents. Ultimately, this project demonstrates the potential and limitations of current deep reinforcement learning methods for real-time driving tasks and establishes a basis for future work in applying more sophisticated algorithms or different learning approaches in simulated environments.

1.3.1 Report Structure

This report will reflect the full lifecycle of the project, integrating technical development with reflective analysis. It begins with foundational context and research framing, followed by a review of relevant literature to situate the work within the broader field. Subsequent chapters cover requirements specification, system design, implementation and evaluation. The structure ensures a logical progression from concept to outcome, supporting both technical clarity and critical reflection on the project's contribution.

2 Literature review

2.1 Artificial Intelligence

Artificial Intelligence is a field of computer science dedicated to creating systems that can perform tasks typically requiring human intelligence, such as reasoning, learning, problem-solving, perception and language understanding (Russell & Norvig, 2003). The foundational goal of artificial intelligence is to develop machines capable of autonomous behaviour and intelligent decision-making, often in environments characterised by uncertainty or complexity.

Early efforts in artificial intelligence focused on explicitly encoding expert knowledge into systems, but these approaches struggled with scalability and adaptability in real-world applications (Nilsson, 2010). A rise in data-driven approaches created a shift towards learning-based models, particularly those that can adapt and improve from experience without explicit programming.

The shift gave rise to the subfield of machine learning, which enables computers to learn patterns and make decisions based on data. Machine learning now forms the foundation of many modern artificial intelligence systems, powering applications from speech recognition and computer vision to autonomous vehicles and intelligent control systems.

2.2 Machine Learning

Machine Learning focuses on developing algorithms and models capable of learning patterns and making data-driven decisions with minimal human intervention. Unlike traditional rule-based systems, which require explicit programming for every scenario, machine learning modules generalise from examples, adapting to new data and evolving as more data becomes available.

Operating on the principle that systems can improve their performance on a task through experience, machine learning is formalised as the ability of a program to learn from data to prove on a task while measuring performance (Mitchell, 1997). Machine learning techniques can be generally classified into three categories: supervised learning, unsupervised learning and reinforcement learning. In supervised learning, models are trained on labelled datasets to predict outcomes, such as recognising handwritten data or detecting fraud in transactions. Unsupervised learning identifies hidden structures or groupings within unlabelled data, as commonly seen in clustering customer behaviour for marketing purposes. Reinforcement learning diverges from these with a trial-and-error approach, where agents interact with an environment and learn optimal behaviour based on rewards and penalties, making this technique particularly well suited for dynamic decision-making tasks like robotics, gaming or autonomous driving (Sutton & Barto, 2018).

The recent resurgence of interest in machine learning has been fuelled by advances in computational power, access to large-scale datasets and the development of more complex architectures, such as neural networks. The field of deep learning uses deep neural networks to extract hierarchical features from raw data, enabling significant performance improvements in domains such as computer vision, natural language processing and speech recognition (LeCun et al., 2015).

These capabilities make machine learning well suited to tasks requiring real-time perception and decision-making, where learning from large streams of visual and behavioural data is essential for effective control.

2.3 Reinforcement Learning

2.3.1 Overview

Key Components of Reinforcement Learning	
Agent	Decision making component
Environment	What the Agent interacts with
State	Representation of the environment at a given time
Action	Decision made by the agent
Reward	Feedback from the environment after an action is taken
Policy	Strategy used to determine actions
Value Function	The expected value returned from an action in a state

Table 1 - Reinforcement Learning Definitions

Reinforcement learning is a machine learning paradigm in which an agent learns to perform optimal behaviours through trial-and-error interactions with an environment, in order to maximise a reward value. Tasks that operate under conditions of uncertainty and require sequential decision making are well suited to reinforcement learning, unlike supervised learning, which relies on labelled datasets to train the agent in making predictions.

2.3.2 Important Algorithms

When an agent has been able to learn the model of the environment it operates within or has direct access to it, it is referred to as a model-based reinforcement learning algorithm. These algorithms try to predict how an environment works using a transition function, which predicts what the next state will be, following an action carried out in the current state. Through the transition function, a model-based algorithm can augment its experience with its environment with simulated experiences, enhancing data efficiency and policy learning.

A suitable task for model-based reinforcement learning could be a robotic arm placing an object in a container, as it can determine the size and the location of the object and container, as well as the physics of its environment. Once these values are learned, it can use the learned model to plan motion trajectories before executing them, reducing the need for trial-and-error with a probabilistic ensemble trajectory sampling algorithm (Chua et al., 2018).

A model-free algorithm learns directly from experience and makes decisions without explicitly modelling the environment's dynamics. Unlike a model-based algorithm, it does not attempt to estimate a transition function, instead focusing on learning a policy or value function.

Figure 1 - Atlas by Boston Dynamics (Amadeo, 2024)

Policy-based methods learn a parameterised policy that directly maps states to actions, optimising the policy by maximising the expected reward using gradient ascent (Williams, 1992). This method is well suited to continuous or multi-model actions spaces, an example of which could be a robot learning to walk on uneven terrain. A policy network would map the robot's sensor inputs to the motor torques, where the robot receives reward signals based on forward velocity, stability and energy efficiency. A proximal policy optimisation algorithm could be used to train the policy, improving model stability and sample efficiency by limiting step sizes in the policy space (Schulman et al., 2017).

Agents that learn to estimate the cumulative future reward from their actions follow value-based reinforcement learning. The methods policy acts greedily with respect to the learned value estimates, using an action-value function which produces a Q value. This method is the basis of deep q-networks, which use deep neural networks to approximate Q values. Deep q-networks use an experience replay, which stores transitions in a replay buffer and samples mini-batches to break correlation in data. A separate target network is then slowly updated over time, improving stability in learning. Epsilon greed exploration balances exploration and exploitation by taking random actions, with the probability of epsilon. These networks are very effective when the number of possible actions is limited, examples of which were first introduced by DeepMind, who used a deep reinforcement learning model to play Atari 2600 games at human-level performance, using the 210 x 160 60Hz video as input (Mnih et al., 2013).

2.4 Autonomous Vehicle Simulation

2.4.1 The Importance of Simulation in Research

Simulations play a crucial role in the development and testing of autonomous vehicles, providing a safe, scalable and cost-effective alternative to real world trials. As self-driving systems must be validated across a vast array of traffic scenarios, weather conditions and road types, real world testing alone becomes impractical and ethically problematic. Simulated environments allow researchers to create and repeat such environments with high fidelity, enabling the development of robust planning and prototyping, without the need to risk human life or property.

The access to simulated environments particularly benefits reinforcement learning models, as agents are able to explore and learn from large numbers of virtual scenarios in extremely short amounts of time; a process that would not be possible in the physical world. Reproducibility and benchmarking are made possible through simulation, allowing researchers to share and evaluate algorithms under consistent conditions. The synthetic data created in these simulations can be used to augment the real-world data collected by autonomous vehicles, further saving time and improving overall performance.

2.4.2 Simulation Platforms

Autonomous vehicle research is often conducted using simulators, which can provide controlled virtual environments to test vehicle and model performance. Companies that develop self-driving vehicles may create their own simulators and systems, such as Tesla, who have developed their Dojo System, which uses specialist design AI training chips to optimise the performance of their neural networks (Tesla, 2025). Other businesses or research teams may choose to work with another company's closed-source services, such as Nvidia's NVIDIA DRIVE Sim, which is said to "support autonomous vehicle development and validation from concept to deployment, improving developer productivity and accelerating time to market" (Nvidia, 2025).

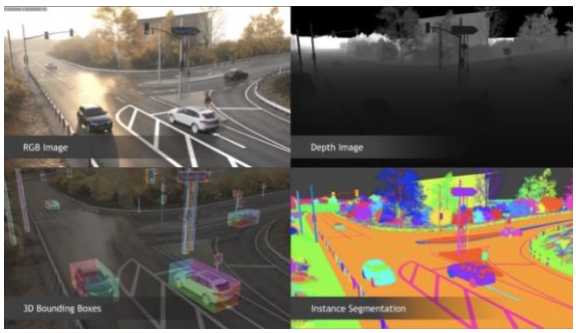


Figure 2 - NVIDIA Drive Sim (Cragun, 2021)

For those without access to significant amounts of money and resources, there are free alternatives for autonomous vehicle simulation. The Massachusetts Institute of Technology ran a competition as a part of one of their deep learning modules, known as DeepTraffic. This competition tasked users with building a neural network and optimising its hyperparameters, in order to navigate an autonomous agent through dense highway traffic. The agent in this competition had the choice of 5 actions: accelerate, brake, change lanes left, change lanes right or do nothing. The simulator showed a top-down two-dimensional view of a 7-lane highway and was written using CovNetJS, running entirely in the web browser. Following the conclusion of the competition in 2019, Lex Fridman, the creator of the competition, published a paper showing how crowdsourcing can be used to tune the hyperparameters of a machine learning model (Fridman et al., 2019).

CARLA (Car Learning to Act) is an open-source simulator based in the Unreal engine, developed at The Autonomous University of Barcelona and supported by the likes of Intel, Nvidia and Toyota. The simulator offers high-fidelity urban environments, with control over weather, lighting and traffic, along with a sensor suite with configurable RGB cameras, LiDAR and radar. CARLA's environment can be accessed by a Python API, allowing for the use of deep learning libraries for autonomous vehicle development. Ego-vehicle control and non-player agents are both supported in the simulator, allowing for the creation of dynamic and interactive traffic scenarios. There are a variety of pre-built maps and assets offered by CARLA, who also offer the option to import external maps, or create custom maps in their RoadRunner plugin (Dosovitskiy et al., 2017).

The high quality and customisable nature of the environments in CARLA make it more computationally intensive, compared to the likes of DeepTraffic, which may pose issues to smaller scale research projects.

2.4.3 Simulation-to-Real Model Transfer

The reality gap between the controlled and safe environment of a simulator and the real world can pose significant challenges to direct deployment of autonomous vehicle models. To bridge this gap, researchers employ techniques such as domain randomisation, where simulation parameters are varied during training, to promote generalisation and domain adaptation, which involves aligning feature distributions between simulated and real data through methods like adversarial learning (Chen et al., 2022).

2.5 Ethics of Real-World Autonomous Vehicles

2.5.1 Ethical Concerns in Autonomous Driving

The deployment of autonomous vehicles introduces complex ethical challenges that extend beyond technological feasibility, most notably on decision-making in critical scenarios, such as unavoidable collisions. The "trolley problem" is a thought experiment that poses a scenario in which a trolley is heading toward a group of five people tied down to the train tracks, where the only way to save them is to pull a lever that diverts the trolley to a track with a single person tied down (Thomson, 1985). This problem has been adapted to autonomous driving, raising questions on how vehicles should weigh the value of different lives and interests in split-second judgements (Bonnefon et al., 2016). If an agent is designed to protect the driver as a primary objective, a question is then raised on the accountability and legal responsibility for any accidents involving autonomous vehicles. This is an argument that remains unsettled, with neither manufacturer, software developer nor driver wanting to take responsibility for any harm that could potentially be caused by an autonomous vehicle.

With some declaring data as the world's most valuable resource (The Economist, 2017), there are many data privacy and surveillance concerns relating to artificial intelligence and autonomous vehicles. Artificial intelligence models require vast amounts of data for training, which in the case of autonomous vehicles, includes cameras, GPS location tracking and user input. This raises issues around user consent, data ownership and the potential misuse of personal information by manufacturers and third parties, as the data collected by autonomous vehicles could be leaked or sold to third parties, who could then use the data outside of its intended purpose, such as targeted advertising.

2.5.2 Safety and Regulation

Safety is a critical aspect of motor vehicle development but with over 90% of road traffic accidents attributed to human error, there is only so much protection that can be provided by physical design and engineering (NHTSA, 2015). The introduction of autonomous vehicles could mitigate the number of incidents caused by human error; however, ensuring this safety requires rigorous testing across a plethora of diverse simulated and real-world environments.

The ambiguity over accident liability remains an unsolved issue, but this has not stopped investment coming in from large governing bodies, with the European Commission investing half a billion euros into their "Cooperative, Connected and Automated Mobility (CCAM)" scheme (European Commission, 2024), who hope to ensure public safety without stifling technological advancement.

A clear, enforceable and adaptive regulatory framework is essential for promoting safe and trustable autonomous vehicles. This can be made achievable through defined liability boundaries, mandatory safety benchmarks and an established process for continuous certification as these systems evolve.

2.5.3 Public Trust and Social Acceptance

The societal integration of autonomous vehicles relies on public trust and despite technological advances, many individuals remain sceptical of the self-driving cars and their decision-making ability in high-risk situations. Studies have suggested there is reluctance for older demographics to engage with autonomous vehicles and more anxiety surrounding the subject for those without higher educational qualifications, suggesting that age and education bear significant impact on the perception of autonomous vehicles (Thomas et al., 2015). With the increasing popularity of artificial intelligence, it can be assumed that the coming generations will have increasingly more understanding of the topic, therefor positively impacting their perception of self-driving cars.

2.6 Autonomous Driving and Artificial Intelligence

2.6.1 Systems Overview

For artificial intelligence and reinforcement learning to be applied to vehicles, there must be a way for the models onboard to establish their environment. This is done using LiDAR, cameras and radar, along with GPS and map data for precise positioning. In practice, a model might struggle to make correct decisions based off sensor perception alone, so the use of pre-mapped environments can significantly improve performance, giving the autonomous vehicle access to lane-level detail, such as road curvature, lane boundaries and traffic sign locations. The models can also make use of rule-based decision making to improve efficiency, in situations such as lane changes, merging onto highways and stopping for pedestrians (Kiran et al., 2021).

Rule-based systems are often used for safety-critical scenarios, which follow more deterministic behaviour and produce consistent outputs; for example, an emergency braking system could detect the vehicle's distance away from an object and its speed, then brake or steer accordingly. An artificial intelligence system is not guaranteed to make the same decision in the same time frame, which could have potentially catastrophic consequences in real-world applications. Similarly, specific object detection could be applied to a rule-based system for sign recognition, enforcing road laws on the driver for stop signs or speed limits. While artificial intelligence is suitable for predictive tasks, it is important to have rules that enforce safety and legality.

Figure 3 - Waymo Driver by Google (Waymo, 2020)

The Waymo Driver was a project that was led by Google 2009, which used a fleet of sensor-equipped vehicles to create a custom map of select areas, similar to Google's StreetView. By doing this, they were able to provide the self-driving cars deployed in those areas with a 3D simulated version of their environment, meaning they didn't need to perceive and process the entire environment in real-time (Waymo, 2020). The data collected by these vehicles was then publicly released in the Waymo Open Dataset, with the hopes of accelerating the development of autonomous driving technology (Sun et al., 2020). While this approach mitigates perception uncertainties, it limits scalability to unmapped regions.

2.6.2 Artificial Intelligence Techniques

Computer vision is a critical technology that allows autonomous vehicles to "see" and interpret their surroundings. By processing visual data from cameras, computer vision systems can identify objects, track movement and make real-time driving decisions, making them essential to a vehicle's perception system. Multiple deep learning models can be used to identify objects, such as road signs, where a first model identifies where a sign exists within an image and once identified, the image is processed, cropping out the unnecessary information and enhancing the image to improve classification accuracy. A classification model then predicts the sign type, which can then be passed to an autonomous agent, where a resulting action can be taken if required. Separating the models improves efficiency while introducing modularity, which can allow each model to specialise in a single task and isolate errors (Gupta et al., 2021) (Madani & Yusof, 2017).

Supervised learning can be used to train computer vision models prior to deployment, where the models are given datasets containing large amounts of labelled samples, allowing the model to recognise patterns and associate those patterns with predefined labels. A successful model, once trained, should be able to identify unlabelled data to a good degree of accuracy. Accuracy can be improved by using a dataset with a variety of samples, ensuring that the model has familiarity with different environments. This includes signs that are partially obscured, at different angles and rotations, different weather conditions and different lighting conditions (S. Houben et al., 2013).

Models that extract meaningful patterns from unlabelled data use an unsupervised learning technique. This is particularly valuable when labelled data is scarce or costly to obtain. Clustering algorithms, like K-means, can be used to identify drivable areas of road from a camera input, without the need for extensive labelled datasets, enhancing adaptability to diverse driving conditions (Sahu et al., 2022-12-04).

Agents can use imitation learning to mimic the actions taken in expert demonstrations, rather than trial-and-error. In autonomous driving, this often involves training a model to replicate the data collected from the sensors of a car driven by a human. A model can use a dataset of actions in given states, then use supervised learning to train a policy that maps observations to control outputs like steering angle, throttle and brake pressure.

Imitation learning allows agents to learn complex behaviours without the extensive exploration required by reinforcement learning, making it particularly valuable in real-world driving scenarios. However, models that use imitation learning can experience distributional shift, which occurs when the model makes small deviations from the expected behaviours, leading to unfamiliar states and subsequent poor performance, as the agent is only trained on states provided by the expert demonstrations (Fujimoto et al., 2024).

Autonomous vehicles that train a single model to map raw sensor input to output controls use end-to-end learning. These models learn a holistic policy by optimising the entire process of perception, planning and control into one differentiable function. This paradigm is typically implemented using deep neural networks can be trained in simulation with reinforcement learning, or by learning from large datasets of expert driving behaviour(Bojarski et al., 2016). The lack of modularity in end-to-end learning can introduce challenges in interpretability, data efficiency and safety validation, as the internal decision-making logic isn't very accessible which makes diagnosing and correcting failure cases more difficult than in modular systems. These models require a considerable amount of training data to be able to generalise reliably, as unencountered edge cases can cause issues.

3 Requirements

The aim of this project is to investigate the application of Deep Q-Networks for learning autonomous driving behaviours in a simulated environment. This section outlines the essential system requirements, organised by functional and non-functional categories.

3.1 Product requirements

This project involves the development of an autonomous vehicle agent using a Deep Q-Network in the CARLA simulator. The agent must be capable of perceiving its environment and making decisions that optimise driver behaviour. An end-to-end approach will be implemented, with the agent making decisions in speed and direction, with later iterations able to navigate a highway environment.

The software is expected to serve as a proof-of-concept or experimental platform for testing learning-based control in simulated environments, intended for academia or research in the field of reinforcement learning or autonomous vehicles.

3.2 Functional requirements

At a fundamental level, the system must allow a reinforcement learning agent to interact with a simulated driving environment. This necessitates real-time communication between the CARLA simulator and the Python-based learning agent. The agent must be able to observe the environment's state and select actions based on those observations, while receiving feedback in the form of rewards.

To begin with, the agent must be able to access an appropriate representation of the environment, including data relating to the vehicle's state, velocity, relative position and lane occupancy. These observations will form the input for the neural network, in the form of an image captured by a virtual camera on the front of the agent. These inputs should provide a discrete spatial representation of oncoming obstacles.

In response to these observations, the agent must be capable of issuing control commands to the environment. These include moving to the left or right lanes and adjusting the throttle to increase or decrease speed. The action space must support interpretable behaviour, including the ability to optimally navigate traffic and avoid collisions where possible.

A key requirement is that the agent must be able to learn from experience. To this end, the system must implement a replay memory, storing past transitions in the form of [state, action, reward, next_state] tuples. Training should proceed by sampling batches from this memory to update the Q-network via stochastic gradient descent, using a target network to stabilise learning. This process must be fully integrated within the CARLA simulation loop, allowing for continuous improvement as the agent interacts with the environment.

In order to provide a sufficiently challenging training environment, the simulator must be configured to represent a multi-lane highway, populated by dynamic non-player vehicles. These vehicles must move realistically, creating opportunities for the agent to adjust speed and road positioning. The simulation must allow for analysis and reproducibility of episodes, enabling fair comparison between different model configurations. This may be achieved through fixed random seeds and deterministic environment resets.

Additionally, the system should support varied driving conditions to elevate generalisation. For instance, weather and lighting conditions in CARLA can be adjusted to simulate fog, rain or night driving. While not essential for the primary demonstration of learning, these features are important for testing robustness and represent a valuable secondary objective.

Finally, the system's reward function must be designed in a modular and transparent manner, allowing for experimentation with different reward weightings, such as penalising collisions, encouraging higher speed or incentivising lane changing. Such flexibility is essential for understanding the effect of reward structure on learning outcomes.

3.2.1 Non-functional requirements

Beyond core functionality, the system should meet several quality attributes to ensure usability, reproducibility and extensibility, most predominantly seen in system performance. Training must be feasible within the time and hardware constraints of the project, making use of the CPU and GPU where possible, as to account for the high computational demand of deep reinforcement learning.

Furthermore, the system should follow an object oriented, modular design, with a clear separation between the simulation logic, learning algorithms and data logging. This not only aids debugging and testing but also allows for the project to be extended in future - for example, by replacing the Cognitive Neural Networks used by the Deep Q-Network with improved models or modifying the simulation environment to expand training. Source code must be readable, well-documented and structured to support iterative development.

During evaluation, the trained agent must be able to operate in real-time or as close to real-time as possible. This allows the agent to respond to observations and issue control commands at a rate that matches the simulation step. This ensures a smooth and believable demonstration of learned behaviour that avoids simulation artefacts that could skew performance.

Finally, the system must be robust to edge cases, including invalid or missing data from the simulator. The learning agent should not crash when confronted with unexpected observations and must gracefully handle episode terminations due to collisions or timeouts, providing feedback to the user in the client.

In summary, these functional and non-functional requirements provide a roadmap for implementing a system capable of demonstrating deep reinforcement learning for autonomous driving in a simulated highway environment. They ensure alignment with project goals and create a solid foundation for subsequent design, implementation and evaluation.

3.3 Design constraints

The development of this project has been shaped by a number of practical and logistical constraints, which have influenced both the design decisions and the scope of implementation. These constraints, while not directly undermining the project's objectives, have imposed limitations that are important to acknowledge in the context of evaluating performance and outcomes.

One of the earliest and more significant constraints was the unexpected departure of the project supervisor during the formative stages of development. This resulted in a temporary gap in academic guidance, particularly during the critical period of project refinement and research. As a consequence, some earlier decisions, such as pivoting from the original DeepTraffic-based plan to the CARLA simulator, were made without the benefit of consistent supervisory feedback that was offered from the first supervisor, who proposed and led the initial project. Although this change ultimately strengthened the project by allowing for greater realism and flexibility, it also introduced additional technical complexity and required time-consuming reorientation of the project's direction.

Another constraint arose from institutional approval processes relating to the access of necessary software and hardware resources. In particular, delays in the confirmation of the new project proposal and obtaining permission to install CARLA and the associated dependencies on university systems. This slowed down initial development and limited the ability to conduct early targeted research and experimentation on the new system. Moreover, as CARLA is resource-intensive and relies heavily on GPU acceleration for rendering and real-time interaction, as with the training of reinforcement learning models, hardware availability posed an additional bottleneck.

A related issue stemmed from version incompatibilities between CARLA, commonly used Python libraries, such as TensorFlow and the available hardware. The versions of CARLA and the Python libraries that were stable and supported the project infrastructure, required the use of downgraded and deprecated library versions, making it difficult to integrate modern GPU-supported systems for the training of the neural network. As a result, the project has to rely on CPU-based training, which significantly increased the time required for experimentation and limited the number of retraining cycles that could be conducted within the project timeline.

These constraints have necessitated compromises in implementation. For instance, hyperparameters within the network had to be modified to improve computational performance, at the expense of model performance, in areas such as replay buffer size, image input size and perceived framerate.

TensorFlow version 1.14.0 requires CUDA version 10.0 or below to leverage the GPU, which was not possible with the available hardware, being an NVIDIA RTX 3070 on Windows 11, as CUDA drivers below version 11.0 are not supported on the operating system. This hinders the system's ability to make real-time decisions, due to the computational intensity of the training and simulation process. While these choices align with the core goals of demonstrating autonomous learning, they inevitably limit the depth of empirical analysis and complexity of the behaviours learned by the agent.

Despite these challenges, the design of the project has prioritised modularity and adaptability, allowing for future iterations to capitalise on improved resources. In this way, the work remains a valuable foundation for further exploration, even if the current implementation reflects some of the limitations imposed by these design constraints.

4 Design

4.1 Network Architecture

The Deep Q-Network is a model-free reinforcement learning algorithm that approximates the optimal action-value function using a neural network. The primary object of a DQN is to learn a policy that maximises the expected cumulative reward over time, by estimating the Q-values for each possible action in a given state.

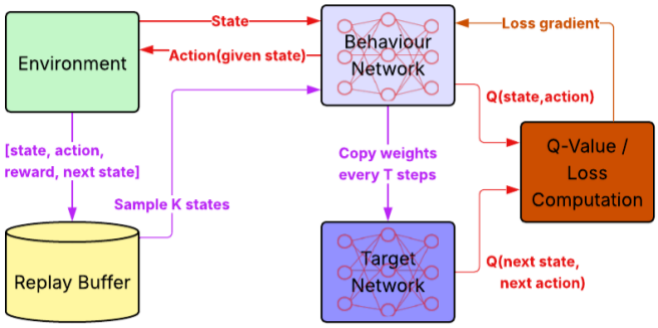


Figure 4 - Deep Q-Network Architecture

```
1:  initialise Q[num_states, num_actions]
2:  observe initial state s
3:  repeat
4:        select and carry out an action a
5:        observe reward r and new state s'
6:        Q[s,a] = Q[s,a] + learning rate(reward+discount
                                  *max(a')Q[s',a']-Q[s,a])
7:        S = s'
8:  until terminated
```

Figure 5 - Q-Learning Pseudocode

The architecture starts with an input layer that receives the current state of the environment. This input is passed through a series of convolutional layers, which extract spatial features relevant to decision making. These layers are followed by one or more fully connected layers that further process the extracted features and output a vector of Q-values, one for each action that an agent can take. The action with the highest Q-value is selected as the policy's output.

Two networks can be used to stabilise learning, where a behaviour network determines which actions are sent to the environment and a target network predicts optimal Q-values, learning from copies of the behaviour network retrieved in regular intervals. This decoupling mitigates any oscillations that can occur when Q-values are updated with rapidly changing estimates.

The learning process is driven by a replay buffer, which contains the agent's past experiences, being a state, the action taken, the reward received and the next state. During training, past experiences are taken from the replay buffer and are randomly sampled by the behaviour network, which breaks correlations between sequential experiences and improves data efficiency. For each experience in a learning batch, the difference between Q-values produced by each network is calculated, producing a loss gradient which can be passed back through the behaviour network to improve estimates over time.

Once an initial network has been created, its performance should be analysed, and improvements should be identified and implemented. This iterative process will be used to improve overall performance through adjustments in architecture and hyperparameter tuning. An appropriate sensor should provide ample information for the network, such as a camera image or LiDAR. The target and behavioural networks balance resource consumption and model performance, with low learning rates to allow for ample exploration. Additional hyperparameters should be adjusted throughout the implementation process to best optimise performance.

4.1.1 Model Architecture

The neural networks used for the target and behaviour networks should be easily interchangeable, allowing for performance evaluation of different modules in similar simulated environments.

Each network will take their input from a sensor within the environment, which will then pass into a feature extraction layer, which will vary depending on input. Initial models will be developed using image data as an input, where convolutional layers should be used to extract spatial features relevant to the environment. These layers should then be followed by flattening and fully connected layers to aggregate the identified features, using nonlinear activations to learn high-level features. Fully connected hidden layers should be used to process the extracted features, learning complex value functions that map input states to expected cumulative rewards. Then finally the output layer should produce a vector of Q-Values corresponding to the discrete action space. The behaviour network selects the action with the highest Q-Value, while the target network provides the Q-value for training updates.

Both networks will use the Adam optimiser, which adjusts the learning rate adaptively for each parameter. Adam is particularly well-suited to reinforcement learning as it handles noisy gradients and sparse rewards more effectively than traditional stochastic gradient descent. It combines the advantages of momentum and adaptive learning rates by maintaining running averages of the gradients and their squares, leading to more stable and faster convergence in dynamic environments (Kingma & Ba, 2017).

Mean squared error loss was used in conjunction with the Adam optimiser, with the goal of minimising the difference between predicted and target Q-values. The loss function penalises larger errors more heavily, encouraging the model to converge towards accurate Q-value estimates over time. An adaptive learning rate optimisation algorithm such as Adam, compliments this by efficiently adjusting weights based on the first and second moments of the gradients. This combination leads to stable and efficient training, making it a standard approach for value-based reinforcement learning problems.

The models used in this environment will select one of five discrete actions, similar to the action space of the DeepTraffic simulator, being change lane to the left, to the right, accelerate, maintain speed or decelerate.

4.1.2 Xception

The Xcpetion, or “Extreme Inception”, model is a deep convolutional neural network architecture proposed by Francois Chollet in 2017 that maps spatial depth-wise and point-wise convolution separately to improve efficiency (Chollet, 2017). The model uses a three flow system, with the entry flow using convolutional layers and max pooling to extract low level features, followed by the middle flow, which uses a series of depth-wise separable convolution blocks to extract features further, ending with the exit flow, which performs the final down sampling into a global average pooling layer, ending with a fully connected softmax classified layer.

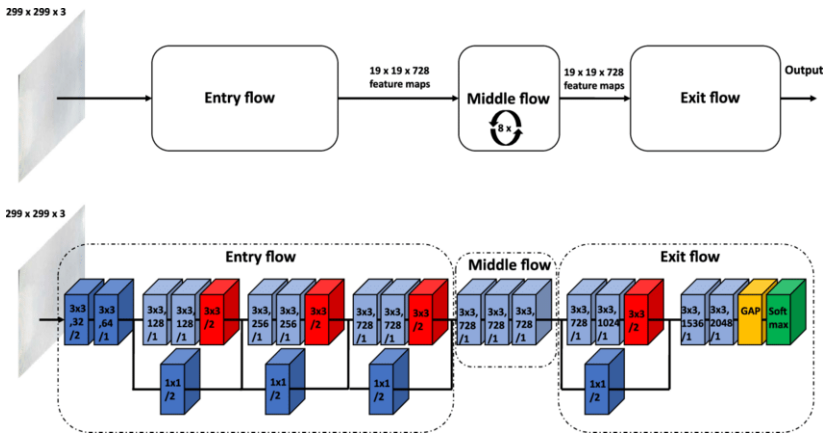


Figure 6 - Xcpetion Model Architecture (Westphal & Seitz, 2021)

A pre-trained version of Xception is available through the Keras library and provides a strong model capable of image processing and classification. The complexity of this model could lead to weaker computational performance on some machines.

4.1.3 Average Pooling (64,64,64)

A more basic convolutional model will be tested in the network, comprised of three convolutional and average pooling layers, each using 64 3x3 filters and rectified linear unit activations to extract features from the image input. These are followed by average pooling layers that reduce the spatial dimensions while retaining important patterns. After feature extraction, the model flattens the feature maps and passes them through a dense layer with 512 neurons to learn high-level representations. The final layer is a linear output layer, matching the number of possible actions.

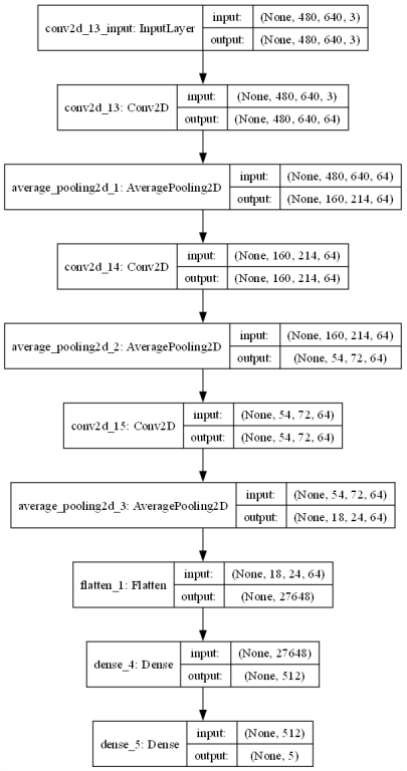


Figure 7 - 64x3 Architecture

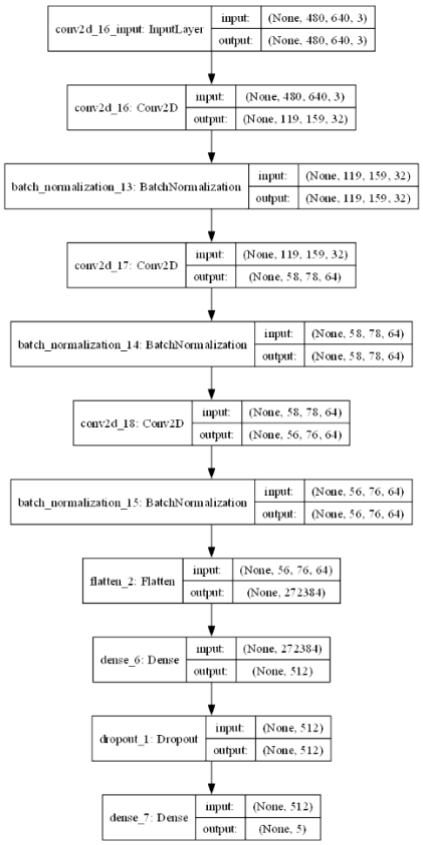


Figure 8 - CNN1 Architecture

4.1.4 Batch Normalisation (32,64,64)

This model also contains three convolutional layers, beginning with a large layer with 32 8x8 filters and a stride of 4, which effectively capture broad spatial features while reducing input dimensionality. This is followed by two additional convolutional layers with decreasing filter sizes, from 4x4 to 3x3, each of which use 64 filters. This progression enables the mode to learn increasingly abstract features across the spatial hierarchy.

Each convolutional layer is followed by Batch Normalisation, which helps standardise the output activations, accelerates convergence and stabilises the learning process. After the feature extraction, the model passes through the same dense layer used in Chapter 4.1.3, followed by a dropout layer with a rate of 0.2, which mitigates overfitting by randomly deactivating neurons during training. The model ends with an output layer with five linear units, corresponding to the number of discrete actions.

Compared to the model in Chapter 4.1.3, which relies on consistent kernel sizes and average pooling to progressively downsample and extract features, this model emphasises more aggressive spatial reduction early on and uses batch normalisation for improved training stability. The inclusion of dropout adds regularisation not present in the other models.

4.1.5 Reward Design

The reward function design is a critical component in shaping the agent's behaviour within the CARLA environment. The objective is to promote safe and efficient driving by balancing positive reinforcement for maintaining optimal speed with penalties for unsafe or unproductive actions. The reward function is manually crafted and should be continuously tuned during training to reflect this goal.

The agent should receive a substantial negative reward for collisions, lane violations and stopping, as these events signify failure. Additional discouragement should be applied for low speeds, increasing in severity as the agent gets slower. Maintaining appropriate speeds should yield positive rewards, which should scale to favour efficient progression without exceeding safe velocity thresholds.

The reward scheme is intended to align the agent's incentives with desirable driving behaviours. However, the impact of specific reward values is highly sensitive to the training environment and model architecture. Although this design theoretically encourages correct performance, further refinement, such as adaptive reward shaping, may be required to improve learning efficiency and generalisation.

4.2 Simulation Environment

The environment presented to the agent within the CARLA simulator was designed with reference to the DeepTraffic simulator. The CARLA simulator offers twelve pre-built maps that facilitate a range of driving scenarios, as well as compatibility with RoadRunner by MathWorks, a dedicated tool for creating custom traffic environments (MathWorks, 2025). The process of designing and importing custom maps through RoadRunner requires specialist knowledge and additional resources that extend beyond the scope of this project. As such, the use of pre-built environments was prioritised, with modifications applied as needed to tailor the simulation to the project's objectives.

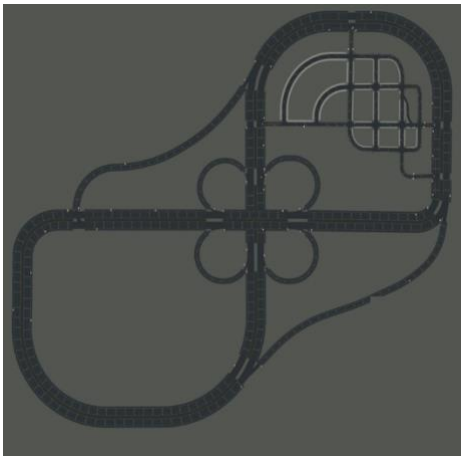


Figure 9 - Town04 (CARLA, 2025)

The "Town04" map hosted a large four-lane highway in a figure-of-8 formation, as well as a small-town area. As the map was relatively small in size and contained long stretches of highway, it was adopted as the map of choice.

CARLA provides the ability to simulate a variety of weather and lighting conditions which can be used to diversify training and test model versatility. Initially, models should be trained in optimal conditions and once they are established and functional, they can be tested in other environments.

Each map includes a set of predetermined spawn points that define where actors can be instantiated. These spawn points are derived from the simulator's underlying waypoint system, which discretises the road network into points placed at regular intervals, containing the appropriate positional and directional data. By constructing an array of waypoints around the agent's initial spawn location, it could be possible to populate the highway with other vehicles or obstacles in a controlled and repeatable manner.

Randomising the positions of surrounding vehicles introduces variability into the environment, which is essential for robust learning policy. This prevents the agent from overfitting to a fixed set of scenarios and encourages the development of generalisable behaviours. By experiencing a wide range of traffic configurations, the agent will learn to adapt its actions to different spatial and temporal contexts, improving its decision-making performance in dynamic and unpredictable real-world conditions.

The agent should be rewarded for maintaining appropriate speed and penalised for any collisions, as to encourage safe and efficient driving behaviours. Collisions serve as a terminal event that resets the environment, returning the agent to the start of the highway. Limiting each training iteration to a maximum of ten seconds ensures that the training process remains computationally efficient and avoids prolonged episodes with minimal learning value. Short episodes also increase the frequency of feedback the agent receives, which will accelerate learning. This design choice balances exploration of the environment with optimal policy evaluation.

Figure 10 - DeepTraffic Simulator (mljack & Fridman, 2018)

4.3 Performance Evaluation

The performance evaluation framework will be developed with a focus on code readability, modularity and real-time feedback to facilitate iterative development and effective model assessment. Code will be well commented and modular, allowing for key components, such as environment configuration, agent architecture, reward logic and training routines, to be independently modified and tested. This design will support debugging and enable the interchangeability of modules for focused experimentation and efficient troubleshooting.

The framework should support real-time analysis tools, such as live sensor readings, the agent's current action and Q-value estimates, and other relevant internal metrics. This feedback will assist in the identification of issues within the learning process, providing valuable insight into the agent's behavioural patterns, decision-making rationale and overall performance.

The reward values collected during training will be used to identify the average, best and worst performance of the agent. These values can be compared against Q-value weights, which will be recorded at regular intervals to monitor convergence. The weights will show the rate at which the model generalised decision patterns and can be used to identify the impact of architecture depth.

The model should be periodically exported, either at fixed time intervals or upon achieving a high reward value. These saved checkpoints should be able to be loaded within the same simulation environment, enabling comparison, review and further testing, without the need for retraining. This will support the evaluation of the training process, as well as ensure traceability.

Computational performance will also be monitorable, with execution times of key operations measured to assess resource efficiency. Collectively, these design decisions will contribute to a robust and transparent evaluation pipeline suitable for reinforcement learning development.

5 Implementation

5.1 Understanding Machine Learning

Machine Learning is a subfield of artificial intelligence that enables systems to identify patterns and make data-driven decisions without explicit rule-based programming. It encompasses several paradigms, including unsupervised learning, supervised learning and reinforcement learning, each suited to different problems depending on the structure of the data and nature of the task. The foundational understanding of machine learning applied to this project was developed through Adil Khan's Machine Learning module at The University of Hull, which focused on the design and training of neural network architectures in image classification tasks.

The module provided practical experience in the application of Convolutional Neural Networks with the Triple MNIST dataset, containing 100,000 84x84 greyscale images of three handwritten digits.

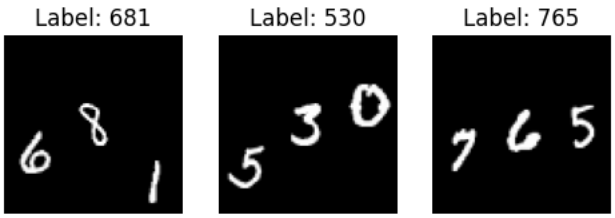


Figure 12 - Random Samples from Triple MNIST

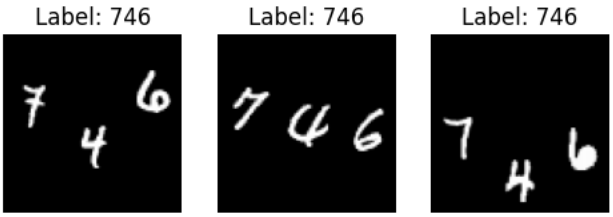


Figure 11 - Variations of Samples with Same Label from Triple MNIST

This task required competence in the training of neural networks and understanding in data pre-processing and model interpretability. Given the complexity of classifying 1000 possible digit combinations, successful models would split the input image into 3 sections, isolating each number, then learn to recognise each number individually, reducing the classification space from 1000 classes to 10. These simplifications significantly improved performance, producing a neural network capable of identifying Triple MNIST images at 97% accuracy.

The skills developed thoughout the module, ranging from image pre-processing, data augmentation, model evaluation and iterative optimisation, have proven directly transferable to the CARLA-based Deep Q-Network project. The experience gained provided valuable practical experience, necessary for designing, implementing and iteratively refining learning-based control systems in complex simulated environments.

5.2 DeepTraffic Simulator

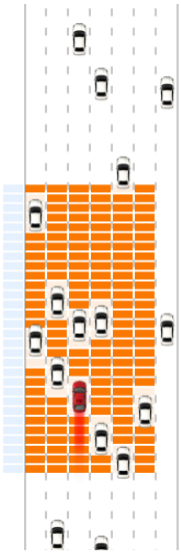


Figure 13 - DeepTraffic Simulator with Learning Input Overlay

DeepTraffic is a browser-based simulation platform developed at the Massachusetts Institute of Technology that enables users to train Deep Q-Network agents to navigate a congested multi-lane highway. The goal for the agent is to drive as fast as possible while avoiding collisions and making intelligent lane changes in real-time. Unlike vision-based environments, DeepTraffic represents the world as a discrete occupancy grid centred around the agent, capturing the positions and velocities of surrounding vehicles within a defined spatial window. This abstraction allows the reinforcement learning algorithm to focus solely on strategic decision-making, without needing to process raw image data.

The initial proposal for this project aimed to evaluate the performance of publicly available models used in the DeepTraffic simulator. This included investigating the architectures and hyperparameters that contributed to optimal agent performance. Preparation for using this simulator involved a review of the associated MIT lecture content led by Lex Fridman, which provided theoretical grounding in reinforcement learning concepts and their application within the DeepTraffic environment (Fridman, 2017).

Although the simulator had been inactive for several years, access was re-established through "mljack" on GitHub, where they hosted a version of the simulator. However, attempts to reproduce previously reported results proved unsuccessful. An evaluation function was used to determine model performance, which would run the trained model through 500 evaluation runs and take the average speed of the agent. Loading models created by other users often resulted in a rapid decline in the "average reward" graph displayed on the simulator, and recreating and training these models again saw the same results. The highest performing models are recorded to have achieved speeds above 75mph but when loading those same models, the simulator failed to evaluate a single model above 70mph. This inconsistency indicated potential version incompatibilities or changes in the simulator's underlying mechanics.

Figure 16 - Reward Graph of 75mph Solution(Parilo, 2017)

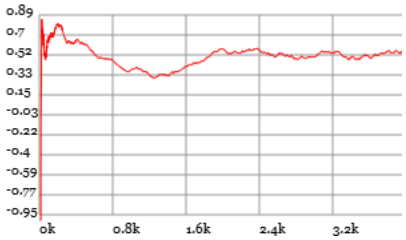


Figure 14 - Reward Graph of Parilo Model in mljack DeepTraffic Simulator

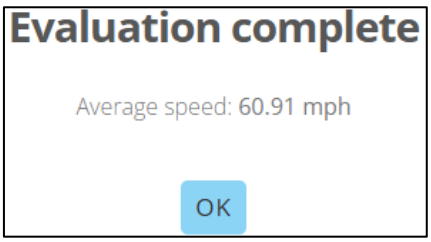


Figure 15 - Evaluation of Parilo Model in mljack DeepTraffic Simulator

The source code of the simulator was analysed, both from the GitHub repository and from scraping the code from a web archive of the original simulator. While this yielded a deeper understanding of the simulator pipeline, reliable replication remained unachievable.

Due to the persistent reproducibility issues and limited scope of the DeepTraffic simulator, the project was redirected toward the implementation of a Deep Q-Network within the CARLA simulation environment. This transition offered greater control on an actively supported platform.

5.3 Environment Setup

To begin working with CARLA, the simulator was locally installed and configured with the appropriate dependencies for Python and the Unreal Engine, followed by the creation of a basic Python-based client which accessed the CARLA API. A foundational understanding of the simulator was built by following the “First Steps” tutorial, found on their website along with their setup recommendations, which introduced key concepts such as spawning objects, accessing the world state and configuring sensors.

To ensure a controlled and effective training environment, a series of scripts were developed to optimise agent positions and environment configuration within CARLA. A preliminary script, find_spawn.py, was used to iterate through CARLA’s predefined spawn points, as to find the optimal location to initialise the agent at the start of each episode. Building on this, a second script, spawn_spectate.py, was created to configure and observe the environment during development and early model testing.

Figure 17 - Town04 (Spawn 170)

Figure 18 - Town04 (Spawn 50)

Figure 19 - Town04 (Spawn 316)

The script enabled direct interaction with the environment and facilitated familiarity with CARLA’s waypoint system. A collection of functions was developed, allowing for the visualisation of waypoints, isolation of individual lanes and calculation of appropriate spacing for spawning vehicles and objects. Iterative testing between environment configuration and model deployment allowed for continuous refinement of vehicle placement logic and overall environment behaviour.

To improve simulation performance and traffic flow, several practical optimisations were introduced. A test vehicle was spawned with autopilot enabled at initialisation to prime the traffic manager. Before vehicle destruction, autopilot must be disabled to prevent deletion issues.

Initial attempts to spawn vehicles randomly across all waypoints on the highway led to clusters that blocked agent movement. To address this, the waypoint space was structured into rows ahead of the agent, limiting vehicle spawning to one per row, thereby improving realism and enabling dynamic overtaking behaviour.

Figure 20 - Waypoint Visualisation

Figure 21- Random Highway Population

Figure 22 - Random 1-Per Row Highway Population

This setup also provided insights into CARLA’s lane management system. Lane identifiers and lane-changing logic were later incorporated into both the action space and the reward function, ensuring that off-road lane changes into the central reservation or hard shoulder would be penalised. The setup phase therefore played a valuable role in constructing a stable, realistic and modular highway driving environment for the training of reinforcement learning agents.

5.4 Deep Q-Network Implementation

5.4.1 Initial Setup

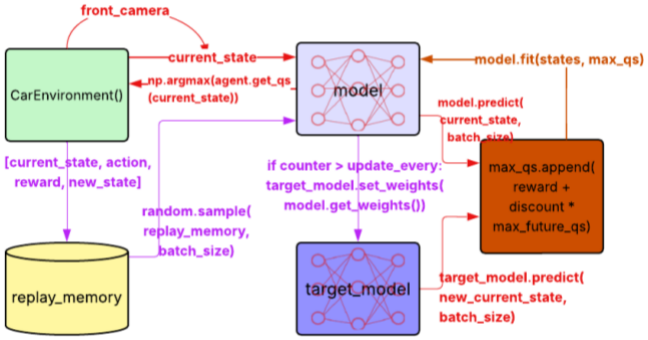


Figure 23 - Deep Q-Network Implementation Architecture

The Deep Q-Network implemented in this project is based on an open-source tutorial developed by Harrison Kinsley (Sentdex) of pythonprogramming.net, which provides a step-by-step video and text-based walkthrough for building a Deep Q-Network in Python using TensorFlow and Keras (Kinsley, 2019). The tutorial focuses on learning control via image-based inputs and discrete actions. This implementation served as a valuable resource for understanding the structure and components of a functioning Deep Q-Network system within Python and the CARLA simulator.

While the foundational structure of the network, including experience replay, target network updates and an epsilon-greedy policy, was retained, several components were adapted to fit the requirements for this project.

The output actions on Kinsley's network modified the steering angle of the agent while maintaining a constant speed. After 10,000 episodes, the model continued to struggle navigating its environment, making it unsuitable for this project.

The modified model mimics the output actions used in the DeepTraffic simulator, being change lane left, change lane right, accelerate, brake and maintain speed. The lane changing functionality made use of the CARLA engine to directly reposition the agent in the adjacent lane, as the task of learning optimal road position and lane changing manoeuvres was not a primary objective of the project.

The change in output action also meant a modification to the reward function and Q-value weighting, which were modified throughout the project.

The network was initially built using the Xception model, which uses depth-wise separable convolutions, allowing it to extract complex visual features efficiently while reducing computational overhead. The model contains pretrained layers, providing a strong foundation for visual understanding, which then pass into fully connected layers that predict Q-values for each output (Chollet, 2017). Functionality was added to replace Xception in place for other Convolutional Neural Networks, which could be tested, developed and swapped throughout the implementation process.

The network was originally designed to leverage the GPU, running training loops as separate threads. However, due to the compatibility issues mentioned in Chapter 3.3, this functionality was limited to the CPU.

```
import tensorflow as tf
from tensorflow.python.client import device_lib
print(device_lib.list_local_devices())
```

✓ 4.0s

```
[name: "/device:CPU:0"
device_type: "CPU"
memory_limit: 268435456
locality {
}
incarnation: 13977443086622288624
]
```

Figure 24 - Available TensorFlow Devices

The reward function applied severe punishment for collisions, with slightly less severe punishment for switching to off-road lanes and a smaller punishment for coming to a stop, all of which end the episode. Low speeds are penalised with increasing severity, with under 10km/h being the harshest. A light reward reduction is issued for speeds under 30km/h, with speeds about gaining reward for safe driving.

5.4.2 Implementation Challenges and Limitations

While Kinsley's Deep Q-Network provided a foundational structure for this project, its implementation revealed a number of critical compatibility and performance issues. The original codebase was built for Windows 10 and relied on legacy versions of TensorFlow and Keras, enabling features such as customising TensorBoard outputs. However, this solution is incompatible with modern Windows 11 CUDA drivers. Consequently, the target system's NVIDIA GeForce RTX 3070 GPU could not be utilised, forcing all reinforcement learning based computation onto the CPU.

To address this, the architecture was modified to allow for training on CPU threads. However, this introduced concurrency issues during Q-value retrieval and weight updates, leading to inconsistent behaviour and race conditions. Thread locks were implemented to ensure synchronisation at the expense of a multi-threaded solution, as when using multiple threads, the locking forced them into series and further increased training time. When forced to run training on the main thread, the network suffered from extreme performance bottlenecks, with training steps taking over 30 seconds and the CPU operating at maximum capacity throughout runtime. This rendered the agent unable to interact with the environment at a sufficient frequency for learning.

In response, support was added for alternative, less resource-intensive models to replace the Xception-based architecture originally used. While these adjustments resulted in modest performance gains, training times remained unacceptably high and continued to limit the agent's ability to learn in real-time.

Modifications to the model and system parameters, such as smaller batch sizes, replay buffer capacity and input size were tested with no significant improvement to performance.

5.5 Debugging Features

Flag / Location	Description
PRINT_ACTIONS	Print agent actions
PRINT_NUM_ACTIONS	Print actions per second
PRINT_QS	Print Q-values
PRINT_QS_DIFF	Print Q-value changes
PRINT_TIMES	Print timing for predictions/training
PRINT_TRAINING	Print training timing
PRINT_THREADS	Print thread status
PRINT_REWARD	Print reward details
Episode Statistics	Save episode reward stats
Q-value logging	Log Q-values every 25 episodes (to txt)
Model Export	Save model on reward threshold
export_constants_to_txt	Save config to file

Table 2 - Debugging Functions

A collection of additional debugging and performance review features was added to the main client, allowing for real-time evaluation of model performance and computational strain.

The open-source solution developed by Harrison Kinsley included scripts that could load exported models, enabling episode recreation, which provides the ability to view real-time sensor input and Q-values. Some light modifications were made to accommodate the additional actions, and to view the chosen action and received reward.

The solution also provided a ModifiedTensorBoard class, which provides additional control over logging during training. It allows for custom metrics to be logged at any point, rather than after a ".fit()" call, which is the default TensorBoard behaviour. This modification aggregates all logs into a single file, making visualisation faster and more organised, while improving computational efficiency. Additional metrics such as episode time, actions per second and Q-values were added to the board, providing additional performance evaluation statistics.

5.6 Network Evaluation

5.6.1 Computational Performance

Evaluation of the trained models reveals the impact of hardware limitations on system performance. Table 3 compares the performance of each model when running training operations on the main thread versus a separate training thread. The results indicate a clear divergence in behaviour between the Xception-based model and the two simpler convolutional models. Specifically, the Xception model exhibited worse performance when training occurred on a separate thread. This is likely a result of how resources are allocated during execution. Running computationally intensive tasks on the main thread appears to allow the operating system greater flexibility in distributing the workload across available CPU cores.

Model	Thread	Q-Prediction	Future Q-Prediction	Model Training	Total Train	Action Selection	Actions per Second
Xception	Main	6.04 sec	6.50 sec	34.89 sec	47.49 sec	0.3 sec	0.02
	Training	15.96 sec	12.09 sec	43.24 sec	71.36 sec	1.3 sec	0.02
64x3	Main	0.69 sec	0.71 sec	2.03 sec	3.47 sec	0.05 sec	0.3
	Training	1.06 sec	1.06 sec	2.49 sec	4.68 sec	0.07 sec	30
CNN1	Main	0.67 sec	0.69 sec	3.40 sec	4.76 sec	0.05 sec	0.19
	Training	0.71 sec	0.70 sec	5.45 sec	6.92 sec	0.19 sec	15.68

Table 3 - Model Computation Time Comparison

Table 4 supports this interpretation, showing that all 20 cores reach 100% utilisation during main-thread training. In contrast, models trained on a separate thread, created dynamically during run-time, may be constrained by thread affinity and operating system scheduling, limiting their access from the full range of available resource. This theory is further reinforced by the CPU core utilisation graphs, which display consistent spikes across all cores during training on the main thread. The intermittent troughs in these graphs correspond to the brief reset periods between episodes, where the environment resets and training temporarily halts.

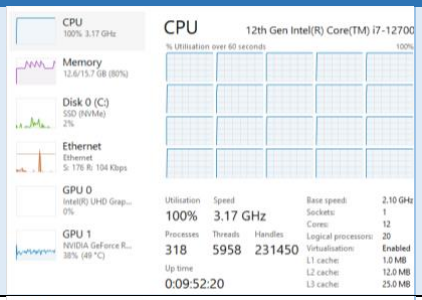
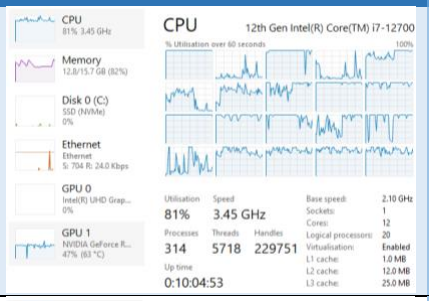
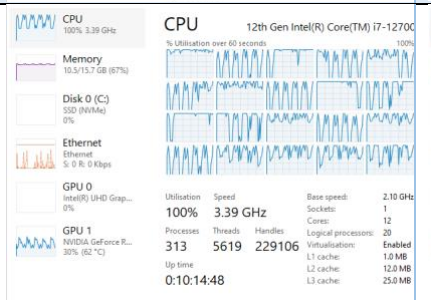
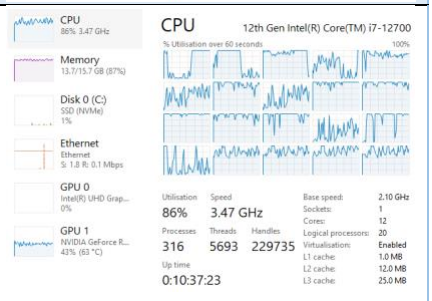
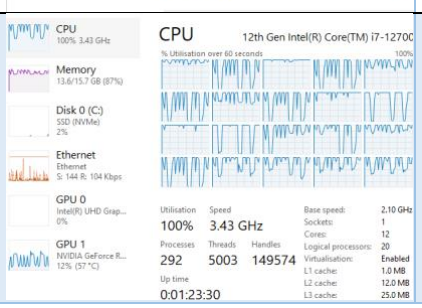
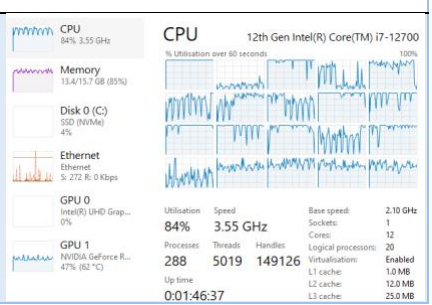
Model	Main Thread	Training Thread
Xception		
64x3		
CNN1		

Table 4 - CPU Utilisation During Training

The observed increase in actions per section for the convolutional models trained on a separate thread does not reflect improved model performance. Despite this apparent improvement, both models continued to exhibit significant latency, with training times resolving far from that of a real-time response. The higher action rate can be attributed to the concurrent nature of the training process. While the training thread updates the model's weights, the main thread continues executing actions; however, during this period, the model is unable to reliably retrieve updated Q-values, which are essential for informed decision-making. The prediction times collected when running on the main thread are consistent with the "time.sleep()" function implemented after selecting a random action, as to allow the environment to update.

```
while True:
    if np.random.random() > epsilon:
        action = np.argmax(agent.get_qs(current_state))
    else:
        action = np.random.randint(0, 5) # random action action_num
        time.sleep(0.05) # wait for random action to be taken
```

Figure 25 - Action Selection Logic

This limitation is particularly evident in the early stages of training, when the epsilon value remains high, and the agent engages in more exploratory behaviour. During exploration, epsilon is closer to 1, meaning actions are more likely to be selected at random, rather than based on Q-value predictions, allowing the system to bypass model inference altogether. These random actions are executed directly on the main thread without the need for model queries, giving a false impression of efficient decision-making. As a result, the concurrency model inflates the number of actions taken per second without corresponding improvements in learned policy effectiveness.

5.6.2 Model Performance

5.6.2.1 CNN1

The inability to train models at a reasonable rate significantly impairs their overall performance. As most models demonstrated similar behavioural patterns, this chapter focuses on a representative run from CNN1, exemplifying the issues with the network. Despite being more lightweight than the Xception-based model, CNN1 still failed to converge towards an optimal policy due to the hardware bottlenecks imposed by the solution.

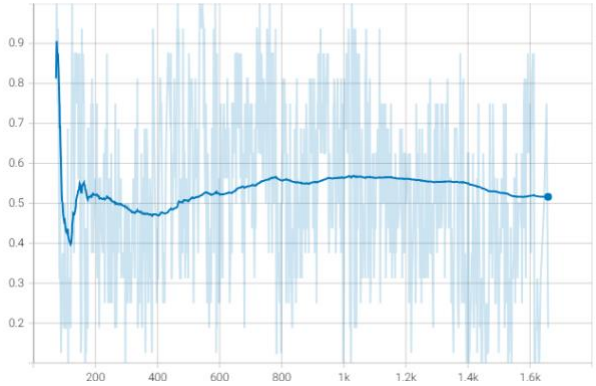


Figure 26 - CNN1 Accuracy

Throughout training, the model's accuracy remained highly unstable, with sharp fluctuations and no discernible upward trend. This lack of consistency suggests the model was not effectively learning from its environment. The instability is likely a result of several compounding factors, such as the delayed or incomplete weight updates and subsequent low frequency of Q-value evaluations.

The lack of GPU acceleration forced all computations onto the CPU, which remained at full utilisation for extended periods. This not only limited training throughput but also reduced the opportunity for the model to refine its policy through experience replay.

The scale of the loss graph in Figure 27 further illustrates the extreme instability of the training process. With the loss reaching values as high as 7e+16, the magnitude of the loss suggests the model is not converging in any meaningful way.

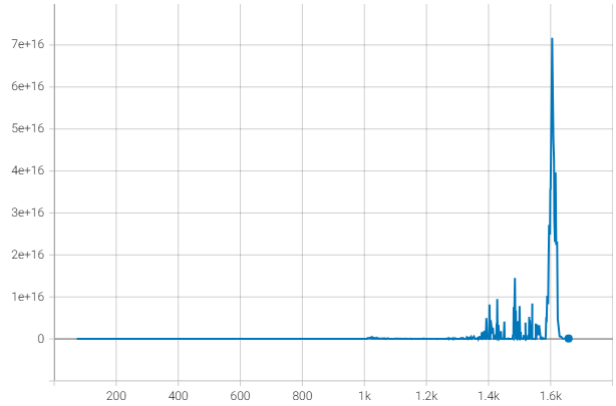


Figure 27 - CNN1 Loss

Figure 28 details the number of actions executed per second, which fails to exceed eight after 600 episodes. This is particularly concerning given that, by this point in training, the epsilon value has decayed substantially, reducing the frequency of randomly exploratory actions that run independently on the main thread. As a result, the model increasingly relies on its learned policy to select actions, as it enters the exploitation phase, yet the observed throughput remains critically low.

This inconsistency in action execution has a cascading effect on the learning process. Irregular action intervals lead to sparse and uneven reward feedback, which in turn introduces instability in the Q-value updates. Without a consistent stream of timely rewards, the model struggles to form accurate value estimations, thereby compounding the problem: erratic Q-values result in poorly informed actions, which further disrupt reward acquisition. This feedback loop renders the agent incapable of meaningful learning, and severely undermines the effectiveness of Deep Q-Network.

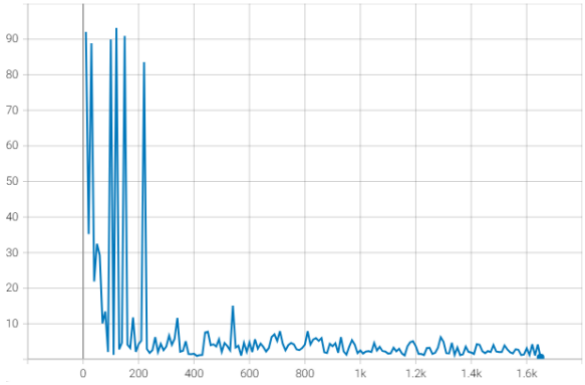


Figure 28 - CNN1 Actions per Second

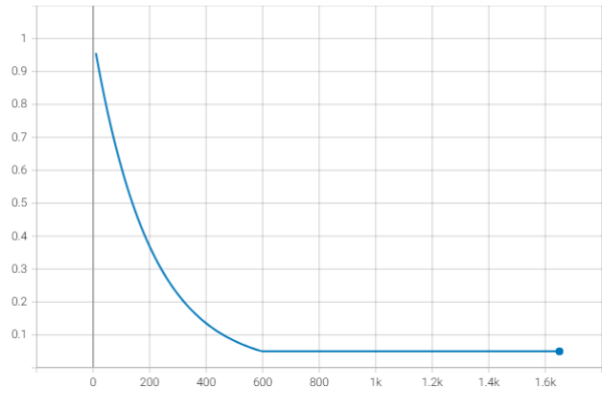


Figure 29 - CNN1 Epsilon

At 1000 steps, both the loss and Q-values exhibit an extreme spike, with such anomalous behaviour strongly suggesting an issue beyond typical model instability or misconfiguration. Instead, the evidence points toward a system-level fault, likely involving resource contention or memory corruption. Given that the model is configured to export its weights every 500 episodes, it is plausible that the export operation at episode 1000 triggered a memory bottleneck or resource starvation event.

Exporting a model involves significant disk I/O and memory usage, especially with large parameter sets or when operating under limited system resources. If the training thread is simultaneously allocating memory to store large tensors while the export function is writing the model to disk, the system could experience heap fragmentation or memory overcommitment. This can result in corrupted memory reads, uninitialised variables or partially written weights being passed back into the model. Such corruption may manifest as nonsensical Q-value predictions, which then feed into the temporal difference loss function. The loss value then becomes unrealistically large, propagating instability throughout the network.

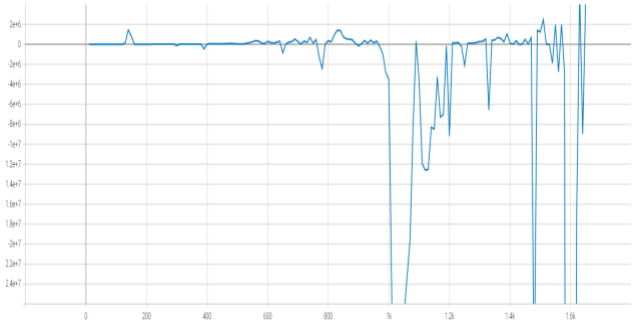


Figure 30 - CNN1 Brake Q-Values

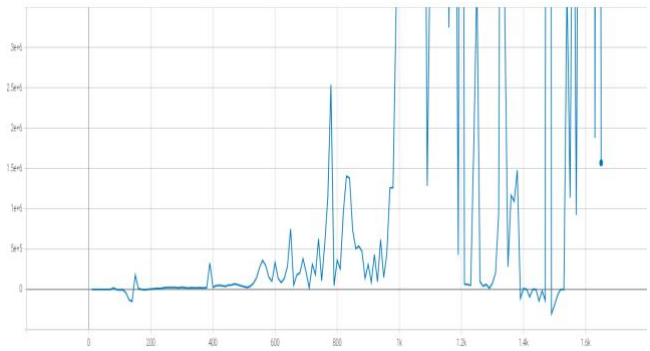


Figure 31 - CNN1 Accelerate Q-Values

5.6.2.2 64x3

A final training run using the 64x3 convolutional model displayed significantly more stable behaviour across multiple evaluation metrics, marking it as the most successful configuration tested. Over the course of the training process, key indicators such as accuracy, loss, average reward and Q-value estimates showed encouraging trends.

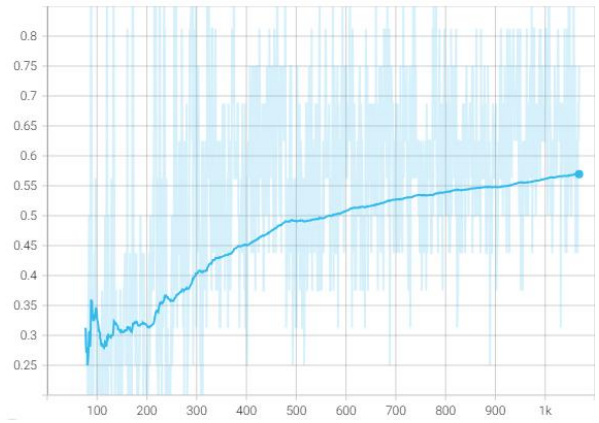


Figure 32 - 64x3 Accuracy

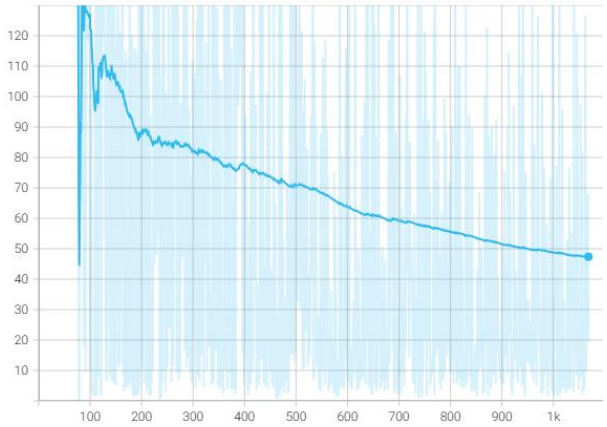


Figure 33 - 64x4 Loss

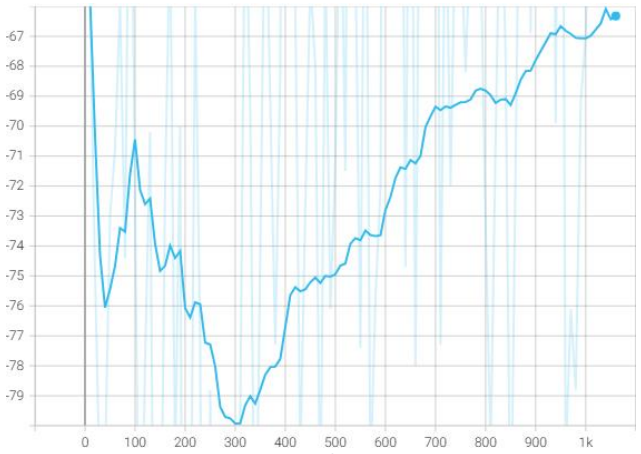


Figure 34 – 64x3 Average Reward

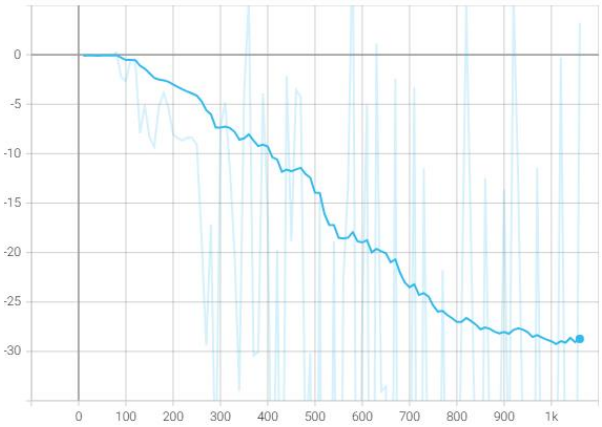


Figure 35 – 64x3 Acceleration Q-Values

Figure 32 shows a steady increase in accuracy, suggesting that the model was able to generalise some understanding of the environment and choose increasingly effective actions. Figure 33 highlights a decline in loss values, indicating improved predictions of expected rewards and more stable convergence of the learning algorithm, all while staying within a suitable range, seen in the y-axis. Moreover, the average reward per episode, depicted in Figure 34, rose gradually and exhibited lower variance compared to other runs, suggesting more consistent interaction with the environment. Perhaps most importantly, the Q-values remained bounded and coherent throughout, as seen in Figure 35.

Although the session was cut short shortly after 1000 episodes due to time constraints, the data gathered strongly supports the potential effectiveness of this model. This run was conducted immediately after the CNN1 experiment discussed in Chapter 5.6.2.1, with a full system restart in between. Importantly, the hyperparameters, reward function and all other core functionalities remained unchanged. This consistency suggests that the improved performance may be attributed to the 64x3 model's slightly less computationally intensive architecture, which could have been better suited to the system's CPU-bound limitations. The observed stability across Q-values, loss and reward trends further reinforces the suitability of this architecture within the current hardware constraints.

6 Evaluation

6.1 Comparison against aims and objectives

The project aimed to investigate the application of Deep Q-Networks in the CARLA simulator, for autonomous driving in a highway setting. The implementation successfully created a simulated highway environment, complete with dynamic non-player traffic and multi-lane behaviour.

However, the project did not fulfil the objective of achieving sustained real-time decision-making. Compatibility issues with TensorFlow and CUDA on Windows 11 prevent GPU acceleration, leading to all computation being performed on the CPU. This limitation significantly hindered the model's ability to train efficiently and respond during simulation steps.

Although multiple neural network architectures were tested and a final model (64x3) demonstrated stable learning behaviour, the agent overall did not consistently exhibit safe driving practices, such as stable lane-keeping and effective collision avoidance. Thus, while the infrastructure and components were correctly implemented, the training outcomes fell short of demonstrating autonomous competence.

6.2 Functional Requirement Evaluation

Requirement	Status	Evaluation
Real-time interaction between agent and CARLA	Partial	Real-time performance was impacted by CPU limitations.
Access to image-based observations and vehicle state	Full	Camera input and vehicle telemetry were successfully acquired.
Discrete action space for speed and positioning	Full	Actions included lane switching and speed control.
Experience replay and target network for training	Full	Used for stable Q-learning.
Simulation of highway environment with realistic traffic	Full	Dynamic traffic successfully simulated.
Support for dynamic training conditions	Partial	Implemented but not evaluated due to focus on core training.
Modular reward design	Full	Reward structure was configurable and adjustable.
Logging and reproducibility	Full	Deterministic resets and model saving supported consistent experimentation.
Real-time interaction between agent and CARLA	Partial	Real-time performance was impacted by CPU limitations.
Access to image-based observations and vehicle state	Full	Camera input and vehicle telemetry were successfully acquired.
Discrete action space for speed and positioning	Full	Actions included lane switching and speed control.
Experience replay and target network for training	Full	Used for stable Q-learning.

Table 5 - Functional Requirements Checklist

6.3 Non-Functional Requirement Evaluation

Requirement	Status	Evaluation
System performance within time/hardware constraints	Partial	CPU-bound training provided limited performance
Object-oriented modular design	Full	Code separates cleanly into agent, simulation and utility components
Real-time response during evaluation	Not Met	Model latency prevented real-time responsiveness
Robust handling of episode terminations and invalid data	Full	Occasional errors can cause handled errors/crashes
Code readability and extensibility	Full	Clear structure and documentation supported by iterative development

Table 6 - Non-functional Requirements Checklist

6.4 Competency and Contribution reflection

The completed work reflects critical and creative problem-solving in adapting reinforcement learning for CARLA, with rigorous experimental design and outcome analysis. Risks including hardware limitations and unstable training were identified and strategies, such as model simplification, were used to mitigate such limitations. Algorithmic choices, such as experience replay, and target networks were selected and critically evaluated against functional and non-functional requirements.

The software was designed with modularity, reliability, and maintainability in mind, particularly under resource-constrained execution. Planning and reflection on time constraints, hardware limitations and technical feasibility ensured realistic goal setting and iteration throughout the development cycle.

The system produced met many of its design and technical goals, with some limitations arising from hardware constraints. It provided a platform for future experimentation and enhancement, while demonstrating the required academic and practical competencies of the degree programme.

7 Conclusion

7.1 Project management

The original methodology and scheduling plans became obsolete following a significant pivot in the project's direction and supervision. Initially developed under a different supervisor with a different scope, the project was redefined in the final third of the allotted timeframe, following growing feasibility concerns of the project raised during the midway review. This resulted in a shift of focus toward a new topic, under new supervision. Consequently, methodological planning had to adapt dynamically to the constraints of a compressed timeframe and altered objectives.

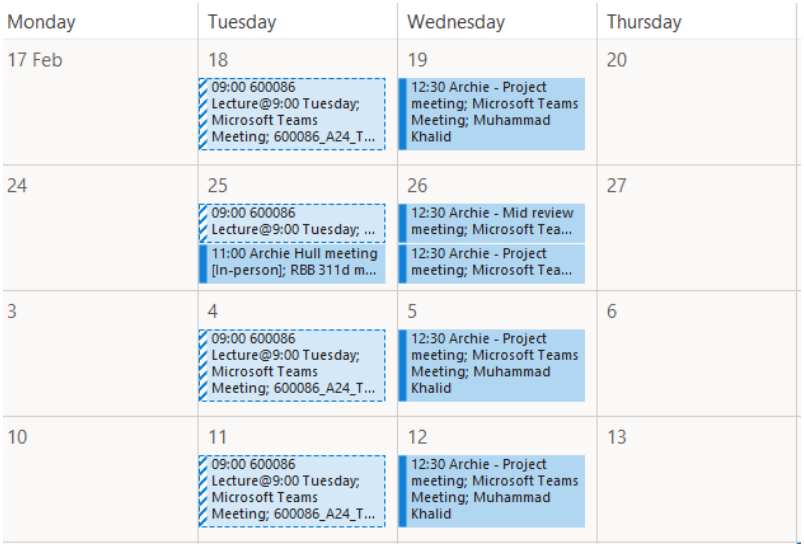


Figure 36 - Calander Showing Weekly Project Meetings

After this pivot, a more agile and iterative approach was adopted. Weekly meetings with the supervisor were maintained, allowing for continuous guidance, regular feedback and incremental development. Progress was monitored informally, rather than through strict adherence to an upfront plan. This pragmatic approach proved appropriate given the evolving requirements and technical challenges encountered.

Discussions in the midway review highlighted the impracticality of the initial topic and helped start the process of refocusing the project around a more manageable and academically valuable objective. This demonstrated the importance of early critical reflection in aligning project ambition with feasibility.

Several lessons emerged from the development process, most notably the importance of flexibility in planning, the value of early risk assessment and the need to critically validate ideas before committing significant development time. The experience also reinforces the benefits of regular evaluation and maintaining realistic goals that can be incrementally refined.

With additional time, further model optimisation could have been undertaken to improve performance under CPU constraints. Additionally, with sufficient resources, the environment could have been adjusted to test a more computationally intensive models by upgrading to newer version of TensorFlow or running experiments on a compatible Windows 10 system.

7.2 Risk management

The original risk analysis and mitigation plan present in Table 7 and the subsequent sections, is revised here. Each identified risk and its corresponding mitigation strategy are evaluated in light of the actual challenges encountered throughout the project.

Risk	Likelihood	Severity	Impact
Loss of access to Deep Traffic Simulator	3	8	24
Data privacy issues	3	7	21
Time management issues	5	5	25
Bugs/errors in code	7	5	35
Damage to personal devices	3	4	12
Ill health	6	2	12
Supervisor departure	2	4	8
Scope creep	4	4	16

Table 7 - Risk Analysis at Initial Project Definition

Loss of access to Deep Traffic Simulation

"As the Deep Traffic simulator is running via a web archiving service, there is a chance for the application to lose functionality or become inaccessible. In this scenario, alternative traffic simulators could be used, which would mean further research would need to be conducted and the code and would need to be modified so it is compatible with the new simulator."

This risk materialised in a more fundamental way than anticipated, as a loss in functionality and project feasibility led to a deliberate abandonment, as the project shifted to a more advanced simulator. The original mitigation plan aligned well with the eventual course of action, although the technical implications of switching to CARLA were underestimated. The risk score accurately reflected its disruptive potential.

Data privacy issues

"This project will be supported by the work of other people, meaning they will need to be appropriately credited. The research conducted in this paper should only consist of publicly available or anonymised data, to avoid any violation of GDPR."

This risk did not materialise as the project used synthetic data generated within the simulation environment. Proper citation practices were maintained, and the planned mitigation was sufficient.

Time management issues

"Timing issues could occur with an increase in workload both in and outside of the project. To mitigate this risk, a task list and timeline had been created to track progress."

Time management was a significant challenge, particularly following the project pivot. The initial Gantt chart and task list became obsolete once the original plan was abandoned. However, post-pivot, a more flexible and iterative approach emerged, allowing for adaptive planning and incremental processes.

Bugs/errors in code

"When implementing reinforcement learning models to Deep Traffic, bugs and errors are likely to occur which will affect model performance and produce inaccurate results. This can be avoided by utilising version control, debugging tools and testing code regularly."

This was accurately identified as the most significant risk and was persistent throughout development. Debugging deep reinforcement learning models in complex simulation environments like CARLA introduced technical challenges. The original mitigation strategies were successfully applied, and the response was effective.

Damage to personal devices
"Personal devices, such as laptops, can sustain damage which may affect functionality. Documentation and code related to this project will be stored on the cloud, meaning if damage occurs that hinders a device's operation, work can be continued on another device, provided there is an internet connection."

This risk did not occur, but use of cloud-based storage and physical backups ensured the work could be recovered if necessary. Any damage sustained to the target machine hosting the CARLA simulator could have potentially raised some issues; however the supporting client code was safely stored in multiple locations.

Ill health
"Illness is likely to occur, however, its effects can be mitigated with planning and time management. Maintaining a healthy lifestyle and following the time sheet and task list will help ensure the project is completed as intended."

No serious health issues interfered with the progress, although the workload at times contributed to stress and fatigue. The risk score likely overestimated the threat posed here.

Supervisor departure
"Project supervisors may depart from the project or university. To reduce the impact of this event, it is important for supervisors and supervisees to keep track of their meetings, allowing for a new supervisor to pick up where the previous left off."

This risk, although scored the lowest, was one of the more consequential. The change in supervisor disrupted continuity but directly contributed to project redirection. However, the increased attentiveness, input and support provided by the new supervisor aided the substantial loss of time. In hindsight, more regular and meaningful interactions with the initial supervisor could have taken place, as well as the discussion and pre-approval of backup projects which could have better addressed this risk.

Scope creep
"The scope of the project may expand as it is written, which can create time management issues and a lack of depth and quality. Clearly defined objectives and tasks, along with regular checks will ensure the project stays within the scope."

Scope creep did not occur in this project, mitigated by regular supervisor meetings, defined technical objectives and the necessity of working in a timely manner.

7.3 General Conclusions

Despite initial setbacks and a pivot from the original DeepTraffic-based plan, the final project made meaningful progress towards its revised objectives. The main contribution of the work lies in the CPU-optimised reinforcement learning pipeline, with ample evaluation and debugging functionality to observe how image-based input can be used to model agent behaviour, using discrete action spaces and modular reward structures.

While the project did not fully meet its aims of achieving real-time autonomous driving behaviour, it successfully established the foundational infrastructure required for further experimentation. The project confirmed that simpler neural network architectures may yield better results than deeper or pretrained alternatives when operating under limited computational resources.

The work enhances the field of autonomous vehicles by contributing a practical, reproducible template for reinforcement learning experimentation in CARLA, particularly under constraints similar to those found in student or small-lab settings. The detailed evaluation of model behaviour, system limitations and stability issues offers insight into common pitfalls when applying deep learning in real-time situations.

The research question has been partially answered. The agent learned some effective behaviours, and the results suggest the viability of the approach, though consistent autonomy and real-time control were not achieved. The failure to do so was primarily due to technical limitations.

7.4 Further Work

With access to more powerful hardware or alternative platforms, more complex models and longer training cycles could be attempted. Algorithmic alternatives might provide better sample efficiency or training stability. Expanding the reward structure to include more detailed behavioural cues and conducting formal hyperparameter tuning would likely improve outcomes. Models could be tuned and tested in different environments and conditions to assess the applicability and versatility.

References

Amadeo, R. (2024, -04-17). *Boston Dynamics' new humanoid moves like no robot you've ever seen.* Retrieved May 13, 2025, from https://arstechnica.com/gadgets/2024/04/boston-dynamics-debuts-humanoid-robot-destined-for-commercialization/

Bojarski, M., Testa, D. D., Dworakowski, D., Firner, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., & Zieba, K. (2016). *End to End Learning for Self-Driving Cars*

Bonnefon, J., Shariff, A., & Rahwan, I. (2016). The social dilemma of autonomous vehicles. *Science, 352*(6293), 1573–1576. 10.1126/science.aaf2654

CARLA. (2025). Maps - CARLA Simulator. https://carla.readthedocs.io/en/0.9.14/core_map/

Chen, X., Hu, J., Jin, C., Li, L., & Wang, L. (2022). *UNDERSTANDING DOMAIN RANDOMIZATION FOR SIM-TO-REAL TRANSFER.*

Chollet, F. (2017, April 4). *Xception: Deep Learning with Depthwise Separable Convolutions.* Retrieved May 11, 2025, from http://arxiv.org/abs/1610.02357

Chua, K., Calandra, R., McAllister, R., & Levine, S. (2018). *Deep Reinforcement Learning in a Handful of Trials using Probabilistic Dynamics Models.* http://arxiv.org/abs/1805.12114

Cragun, M. (2021, -04-12). NVIDIA DRIVE Sim Powered by Omniverse Available for Early Access This Summer. https://blogs.nvidia.com/blog/nvidia-drive-sim-omniverse-early-access/

Dosovitskiy, A., Ros, G., Codevilla, F., López, A., & Koltun, V. (2017). CARLA: An Open Urban Driving Simulator.

European Commission. (2024). *Automated mobility in Europe: where are we now? - European Commission.* Retrieved May 5, 2025, from https://research-and-innovation.ec.europa.eu/news/all-research-and-innovation-news/automated-mobility-europe-where-are-we-now-2024-04-17_en

Fridman, L. (2017, 25 Jan). *MIT 6.S094: Convolutional Neural Networks for End-to-End Learning of the Driving Task.* Retrieved 14/10/2024, from https://www.youtube.com/watch?v=U1toUkZw6VI

Fridman, L., Terwilliger, J. & Jenik, B. (2019, January 3). *DeepTraffic: Crowdsourced Hyperparameter Tuning of Deep Reinforcement Learning Systems for Multi-Agent Dense Traffic Navigation.* Retrieved Jan 26, 2025, from http://arxiv.org/abs/1801.02805

Fujimoto, T., Suetterlein, J., Chatterjee, S. & Ganguly, A. (2024, February 5). *Assessing the Impact of Distribution Shift on Reinforcement Learning Performance.* Retrieved May 4, 2025, from http://arxiv.org/abs/2402.03590

Gupta, A., Anpalagan, A., Guan, L., & Khwaja, A. S. (2021). Deep learning for object detection and scene perception in self-driving cars: Survey, challenges, and open issues. *Array, 10*10.1016/j.array.2021.100057

Kingma, D. P., & Ba, J. (2017, January 30). *Adam: A Method for Stochastic Optimization.* Retrieved May 12, 2025, from http://arxiv.org/abs/1412.6980

Kinsley, H. (2019). Self-driving cars with Carla and Python. https://pythonprogramming.net/introduction-self-driving-autonomous-cars-carla-python/

Kiran, B. R., Sobh, I., Talpaert, V., Mannion, P., Sallab, A. A. A., Yogamani, S. & Pérez, P. (2021, January 23). *Deep Reinforcement Learning for Autonomous Driving: A Survey.* Retrieved May 3, 2025, from http://arxiv.org/abs/2002.00444

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature, 521*(7553), 436–444. 10.1038/nature14539

Madani, A., & Yusof, R. (2017). Traffic sign recognition based on color, shape, and pictogram classification using support vector machines. *Neural Computing and Applications, 30*(9), 2807. 10.1007/s00521-017-2887-x

MathWorks. (2025). *RoadRunner.* Retrieved May 11, 2025, from https://uk.mathworks.com/products/roadrunner.html

Mitchell, T. (1997). *Machine Learning*

mljack, & Fridman, L. (2018). DeepTraffic | MIT 6.S094: Deep Learning for Self-Driving Cars. https://selfdrivingcars.mit.edu/deeptraffic/

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013, December 19). *Playing Atari with Deep Reinforcement Learning.* Retrieved May 2, 2025, from http://arxiv.org/abs/1312.5602

NHTSA. (2015). *Critical Reasons for Crashes Investigated in the National Motor Vehicle Crash Causation Survey*. NHTSA.

Nilsson, N. (2010). *The quest for artificial intelligence: A history of ideas and achievements*

Nvidia. (2025). *NVIDIA DRIVE Sim.* Retrieved May 4, 2025, from https://developer.nvidia.com/drive/simulation

Parilo. (2017, Jan 24). *Parilo - DeepTraffic Solution.* https://github.com/parilo/DeepTraffic-solution

Russell, S., & Norvig, P. (2003). *Artificial Intelligence, A Modern Approach. Second Edition*

S. Houben, J. Stallkamp, J. Salmen, M. Schlipsing, & C. Igel. (2013). Detection of traffic signs in real-world images: The German traffic sign detection benchmark. Paper presented at the *The 2013 International Joint Conference on Neural Networks (IJCNN),* 1–8. 10.1109/IJCNN.2013.6706807

Sahu, N., Chamola, V., & Rajkumar, R. R. (2022-12-04). A Clustering and Image Processing Approach to Unsupervised Real-Time Road Segmentation for Autonomous Vehicles. Paper presented at the 160. 10.1109/gcwkshps56602.2022.10008782

Schulman, J., Wolski, F., Dhariwal, P., Radford, A. & Klimov, O. (2017, August 28). *Proximal Policy Optimization Algorithms.* Retrieved May 2, 2025, from http://arxiv.org/abs/1707.06347

Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., Vasudevan, V., Han, W., Ngiam, J., Zhao, H., Timofeev, A., Ettinger, S., Krivokon, M., Gao, A., Joshi, A., . . . Anguelov, D. (2020, May 12). *Scalability in Perception for Autonomous Driving: Waymo Open Dataset.* Retrieved May 3, 2025, from http://arxiv.org/abs/1912.04838

Sutton, R., & Barto, A. (2018). Sutton & Barto Book: Reinforcement Learning: An Introduction. *MIT Press, Cambridge, MA,* http://www.incompleteideas.net/book/the-book-2nd.html

Tesla. (2025). *AI & Robotics | Tesla United Kingdom.* Retrieved May 4, 2025, from https://www.tesla.com/en_gb/AI

The Economist. (2017). The world's most valuable resource is no longer oil, but data. *The Economist,* https://www.economist.com/leaders/2017/05/06/the-worlds-most-valuable-resource-is-no-longer-oil-but-data

Thomas, E., McCrudden, C., Wharton, Z., & Behara, A. (2015). The Perception of Autonomous Vehicles by the Modern Society: A Survey. *IET Research Journals,* https://research.edgehill.ac.uk/ws/portalfiles/portal/29212693/Main_Revised_Manuscript.pdf

Thomson, J. J. (1985). The Trolley Problem. *The Yale Law Journal, 94*(6), 1395–1415. 10.2307/796133

Waymo. (2020). *The Waymo Driver Handbook: How our highly-detailed maps help unlock new locations for autonomous driving.* https://waymo.com/blog/2020/09/the-waymo-driver-handbook-mapping

Westphal, E., & Seitz, H. (2021). A Machine Learning Method for Defect Detection and Visualization in Selective Laser Sintering based on Convolutional Neural Networks. *Additive Manufacturing, 41*, 101965. 10.1016/j.addma.2021.101965

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning, 8*(3), 229–256. 10.1007/BF00992696