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: XXXXX

This document is a template for your guidance. *You don’t have to stick to it precisely.* It may not suit your particular project. Some section headings may not be relevant to your project, so delete them if you don’t need them. Add other section headings if you need to, but use the predefined styles (see below) to make sure that the table of contents updates itself. Make other modifications if you need to – but please discuss with your supervisor before making substantial changes to the organisation and content.

Paragraphs in red, like this one, are instructions and extra information. You must delete them and the box they are in before submitting your report.

This template document has a number of paragraph styles predefined. If you use them (‘Heading 1’, ‘Heading 2’ and ‘Heading 3’) and don’t alter them, then your report will automatically have properly numbered paragraphs and your table of contents will be automatically generated with the right page numbers. Use ‘Normal’ as the style for general text paragraphs in your document.

The document is divided into sections by ‘section breaks’ in Word. These are needed to keep the page numbering correct, so only delete them if you know what you are doing.

On this page (and I hope this is obvious) you must replace the words “The Title Of Your Project”, “BSc/Beng/MEng”, “Your degree title here” and “Your Full Name Here” with the correct information. You’d be surprised how often people forget to do this.

You must also replace “XXXXX” with the actual word count (excluding acknowledgements, abstract, table of contents, references and appendices) of your document. Note penalties for exceeding the word count will be applied, according to university regulations.

Abstract

The abstract is a short, self-contained statement describing the whole of your work. It should be less than a page (typically half a page) and should summarise the scope, purpose, results and content of the work. The abstract might be thought of as a summary which you would read quickly to decide if the rest of the document is worth taking the time to read in detail. In scientific publishing, abstracts are often used as sources of keywords and concepts for searching, so it’s important to ensure that the main ideas and conclusions of your work are present.

When someone has read your abstract, they should know what your project was about, how you did it and what the end result was. It doesn’t need to contain references or literature reviews. For example, here’s the abstract from Bell and Brooks (2019)’ s paper “*What makes students satisfied? A discussion and analysis of the UK's national student survey*” (which, incidentally, is very interesting):

“*This paper analyses data from the National Students Survey, determining which groups of students expressed the greatest levels of satisfaction. We find students registered on clinical degrees and those studying humanities to be the most satisfied, with those in general engineering and media studies the least. We also find contentment to be higher among part-time students, and significantly higher among Russell group and post-1992 universities. We further investigate the sub-areas that drive overall student satisfaction, finding teaching and course organisation to be the most important aspects, with resources and assessment and feedback far less relevant. We then develop a multi-attribute measure of satisfaction which we argue produces a more accurate and more stable reflection of overall student satisfaction than that based on a single question.*”

Note that it’s very specific. There is no waffle, just details of what they did, how they did it, and what their conclusions were.

Contents

[Abstract i](#_Toc197974546)

[1 Introduction 3](#_Toc197974547)

[1.1 Background to the project 3](#_Toc197974548)

[1.2 Research question 4](#_Toc197974549)

[1.3 Aims and objectives 4](#_Toc197974550)

[1.3.1 Report Structure 4](#_Toc197974551)

[2 Literature review 5](#_Toc197974552)

[2.1 Reinforcement Learning 5](#_Toc197974553)

[2.1.1 Overview 5](#_Toc197974554)

[2.1.2 Important Algorithms 5](#_Toc197974555)

[2.2 Autonomous Driving and Artificial Intelligence 7](#_Toc197974556)

[2.2.1 Systems Overview 7](#_Toc197974557)

[2.2.2 Artificial Intelligence Techniques 7](#_Toc197974558)

[2.2.3 Emerging Technologies 8](#_Toc197974559)

[2.3 Simulation Environments for Autonomous Vehicle Research 8](#_Toc197974560)

[2.3.1 The Importance of Simulation in Research 8](#_Toc197974561)

[2.3.2 Simulation Platforms 9](#_Toc197974562)

[2.3.3 Simulation-to-Real Model Transfer 9](#_Toc197974563)

[2.4 Ethical and Practical Considerations 9](#_Toc197974564)

[2.4.1 Ethical Concerns in Autonomous Driving 9](#_Toc197974565)

[2.4.2 Safety and Regulation 10](#_Toc197974566)

[2.4.3 Public Trust and Social Acceptance 10](#_Toc197974567)

[3 Requirements 11](#_Toc197974568)

[3.1 Product requirements 11](#_Toc197974569)

[3.2 Functional requirements 11](#_Toc197974570)

[3.2.1 Non-functional requirements 12](#_Toc197974571)

[3.3 Design constraints 12](#_Toc197974572)

[4 Design 14](#_Toc197974573)

[4.1 Network Architecture 14](#_Toc197974574)

[4.1.1 Model Architecture 15](#_Toc197974575)

[4.1.2 Xception 15](#_Toc197974576)

[4.1.3 Average Pooling (64,64,64) 16](#_Toc197974577)

[4.1.4 Batch Normalisation (32,64,64) 16](#_Toc197974578)

[4.1.5 Reward Design 17](#_Toc197974579)

[4.2 Simulation Environment 17](#_Toc197974580)

[4.3 Performance Evaluation 18](#_Toc197974581)

[5 Implementation and testing 19](#_Toc197974582)

[5.1 Implementation **Error! Bookmark not defined.**](#_Toc197974583)

[5.1.1 Understanding Machine Learning 19](#_Toc197974584)

[5.1.2 DeepTraffic Simulator 20](#_Toc197974585)

[5.1.3 Environment Setup 20](#_Toc197974586)

[5.1.4 Deep Q-Network Implementation 21](#_Toc197974587)

[5.1.5 Initial Setup 21](#_Toc197974588)

[5.1.6 Implementation Challenges and Limitations 22](#_Toc197974589)

[6 External concerns 24](#_Toc197974590)

[6.1 Legal, social and ethical issues 24](#_Toc197974591)

[6.2 Commercial issues and exploitation 24](#_Toc197974592)

[7 Evaluation and discussion of results 25](#_Toc197974593)

[7.1 Programme specific concerns 25](#_Toc197974594)

[8 Conclusion 26](#_Toc197974595)

[8.1 Project management 26](#_Toc197974596)

[8.2 Risk management 26](#_Toc197974597)

[8.3 General conclusions 26](#_Toc197974598)

[References 27](#_Toc197974599)

[Appendix A – Interesting but not vital material 28](#_Toc197974600)

[Appendix B – Other things which may be useful 29](#_Toc197974601)

# Introduction

In this section you will describe the project’s purpose, aims and objectives. You will introduce the project’s stakeholders and the reason for doing it. You will provide a brief overview of the report’s organisation – which may of course be different to this template.

There may be some overlap with the content of the PDD in this section, but it should not simply be a repeat. The introduction in this report will be informed by the activities you have undertaken and their results, whereas the PDD was concerned with forward planning.

Note that the sub-headings below are suggestions only; you may organise this section differently as appropriate to your project.

## 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 positions 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.

## 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 environment.

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.

## 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.

Academic contribution and achievement, be specifically precise on what I am doing myself from my direct achievement

### Report Structure

# Literature review

The literature review is a survey of the history and state of the art in the domain of your project. It will summarize the work that has already been done in the field; this may be scientific literature, known techniques, and even previous student projects. It will provide a historical perspective on how the subject area has arrived at its current state by looking at important developments over time. If appropriate, it may examine existing software in the domain, especially in terms of the technology used and the features offered. The focus of the literature review is to summarise the existing arguments and ideas of others, identifying which are important.

A good literature review could be a project on its own, and form a very useful guide to anyone new to the particular field. It would identify the important work, authors and publications which would be a good place to begin research activity. Open questions and areas where new work is required would be discussed. Really good reviews are often published in scientific journals. Your literature review is not expected to be quite so substantial, but should still provide a comprehensive summary which will allow the reader to understand the field.

Images can be very useful here. Remember to attribute them properly to avoid accusations of plagiarism. Your literature review will naturally refer to lots of existing work, which must all be properly cited and referenced – see the ‘References’ section towards the end of this document.

Introduction?

Background

What’s happening in background

Hone in on subject, start genereric

## Reinforcement Learning

### Overview

Start with machine learning and move to reinforcement learning

|  |  |
| --- | --- |
| **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 |

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 (Sutton & Barto, 2018).

### Important Algorithms

Applied reinforcement learning, focus on state of the art

Merge with current task

More recent state of art in the subject

Find recent studies that use reinforcement learning and what models, auton driving

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 real-world 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.

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).

## Autonomous Driving and Artificial Intelligence

### Systems Overview

Applications of RL in autonomous systems, then highlight on what I am doing

Combine with next and previous section

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 practise, 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 vehicles distance away from 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 application. 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.

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 an select areas, similar to Google’s StreetView. By doing this, they were able 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.

A screenshot of a video game

AI-generated content may be incorrect.

### 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 vehicles 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 pre-defined 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). If this varied data isn’t accessible, a generative adversarial network could be used to augment the dataset with synthetic data, assuming the output was accurate enough.

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 exportation 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 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 efficient 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 issue.

### Emerging Technologies

Multi model sensor fusion

## Simulation Environments for Autonomous Vehicle Research

### 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.

### Simulation Platforms

Move earlier

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).

For those without access to significant amounts of money and resources, there are free alternatives for autonomous vehicles 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.

### 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).   
read: https://arxiv.org/abs/1703.06907

## Ethical and Practical Considerations

Not related to current project

### 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, an argument 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 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.

### 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.

Current regulations – UNECE, ISO 26262

### Public Trust and Social Acceptance

The societal integration of autonomous vehicles relies on public trust and despite technological advances, many individuals remain sceptical on 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 bare 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 on the topic, therefor positively impacting their perception of self-driving cars.

# Requirements

If your project is primarily concerned with developing software or hardware, then you will be expected to include a section describing the requirements. Some of this might well come from primary research. If so, document it here. If yours is a more theoretically based research project, this section might be ‘Theoretical development’ instead. These requirements will have a basis in the PID’s objectives and deliverables, but they may have changed. If so, explain why.

The suggestions below are not definitive.

What you’re trying to achieve

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-function categories.

## Product requirements

What will your software or hardware do? Who requires it? You might want to refer back to your aims and objectives to inform this section, and perhaps consider if they are still appropriate. UML use case diagrams are very helpful here (even for hardware).

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.

Refer back to previous goals

## Functional requirements

The exact content here will vary (especially if your project is hardware-based), but there are some standard items which you should consider including:

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 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 throttle to increase or decrease speed. The action space must support must, 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 environments 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.

### 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.

## Design constraints

You might include this in the next section if you prefer. Consider the limitations on how you are able to conduct your project. Relate the bounds (time and resources are obvious ones) which have an impact.

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 priorities 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.

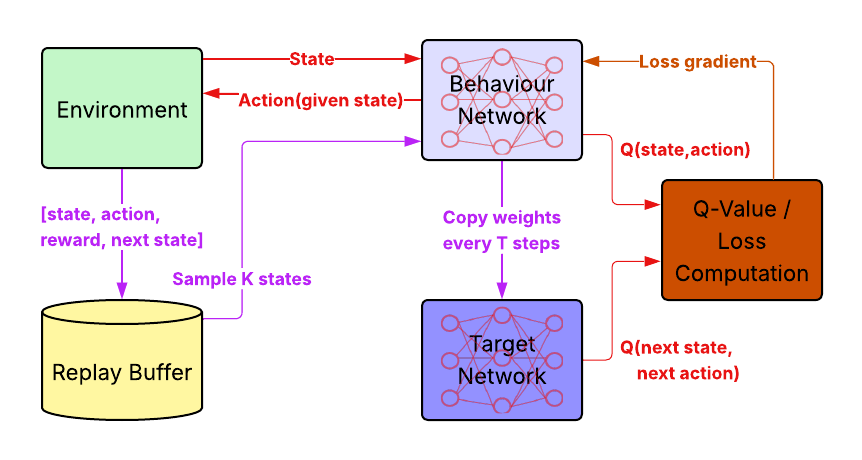
# Design

If your project involves the development of software and/or hardware, then you will need to include a section in which you describe its design in detail. If you conduct any experiments (either in a research-oriented project or simply doing user evaluation) then you should describe their design and methodology here.

Typical content will be detailed software design, from architecture to implementation level. As well as your text, you should include UML diagrams, including class structures, activity and sequence diagrams as appropriate. Don’t just drop diagrams in willy-nilly, though. Use them strategically to illustrate points in your text. Remember that ‘a picture is worth a thousand words’ (we don’t apply this rule literally) but pictures on their own don’t explain everything. If your project requires user interface design, don’t forget to include that. Screenshots, wireframes and other diagrams are welcome. If you are going to evaluate your software or hardware by means of any tests or surveys, then explain their design here. If you are doing other experiments (for example measuring the performance of algorithms, extracting data from environmental monitoring systems or evaluating the performance of mechanisms) then you should explain how you have designed the experiments, how they must be conducted and what you expect to learn from them. This is especially important for research-oriented projects.

## 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.

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.

A computer code with black text

AI-generated content may be incorrect.Two networks can be used to stabilise learning, where a behaviour network determines which actions 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 improving 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 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.

### 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.

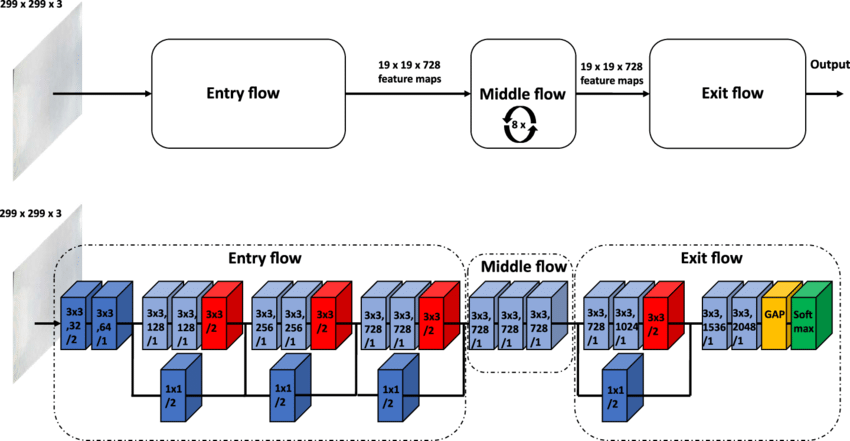
Both networks will use the Adam optimiser, which adjusts the learning rate adaptively for each parameter. Adam is particularly well-suited to reinforcement learning as it handles noisy gradients and sparse rewards more effectively than traditional stochastic gradient descent. It combines the advantages 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 stand 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.

### 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 pointwise 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 later, ending with a fully connected soft max classified layer.



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.

### 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.

### 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 Section 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 Section 4.1.3, which relies on consistent kernel sizes and average pooling to progressively down sample 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.

### Reward Design

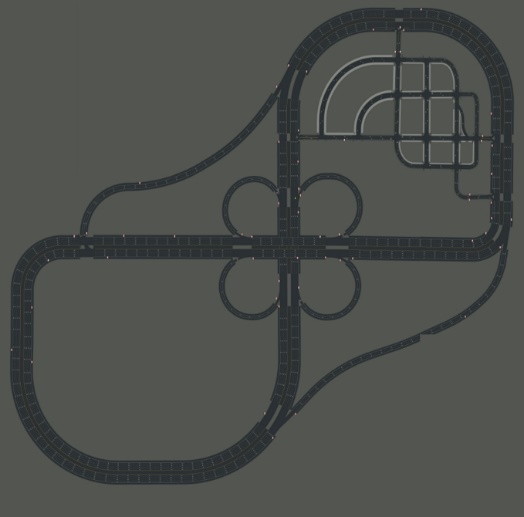
The reward function design is a critical component in shaping the agent’s behaviour within the CARLA environment. The objective is to promote safe and efficient driving by balancing positive reinforcement for maintaining optimal speed with penalties for unsafe or unproductive actions. The reward function 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 with 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.

## 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 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.

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 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 pre-determined spawn points that define where actors can be instantiated. These spawn points are derived from the simulators 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.

A car on the road

AI-generated content may be incorrect.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.

## 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 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 ensuring traceability.

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

# Implementation

In this section you will describe what you did, and why you made the important decisions affecting your actions. It’s not a diary – don’t write a blow-by-blow account of every little thing that happened. Be selective and report those choices and techniques which made a difference. Make sure you discuss what options you considered. Explain how the criteria and methodology you used to select amongst different options (which tools are most appropriate, for example).

It may help to imagine that you are reading this project in the future, trying to replicate the work without making the same mistakes along the way. What would you need to know to make your job easier, and what is unimportant or obvious? Explain how you implemented the design in the previous chapter.

This is the place in which you would explain any novel or especially complex algorithms, data structures or systems you have used.

Make it clear what you have done, and what is pre-existing. For example, if you are using third party software libraries, describe how you have used them, and how they have benefited your project rather than simply what they do. If you have built on a framework, make it clear how you have developed new functionality.

## 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.

A black square with white text

Description automatically generated A black square with white text

Description automatically generated

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 though out the module, ranging from image pre-processing, date 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.

## DeepTraffic Simulator

A map of cars on a grid

AI-generated content may be incorrect.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 leaning 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 eval and take the average speed of the agent. Loading models created by other users often resulted in a rapid decline 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.

A graph with red line

AI-generated content may be incorrect. A graph with a red line

AI-generated content may be incorrect. A screen shot of a computer

AI-generated content may be incorrect.

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.

## 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 access 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 withing 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.

A car on a road

AI-generated content may be incorrect. A car on a road

AI-generated content may be incorrect. A car driving on a road

AI-generated content may be incorrect.

170, 50, 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 to 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 structed into rows ahead of the agent, limiting vehicle spawning to one per row, thereby improving realism and enabling dynamic overtaking behaviour.

A video game of a road with cars and trees

AI-generated content may be incorrect.A screenshot of a video game

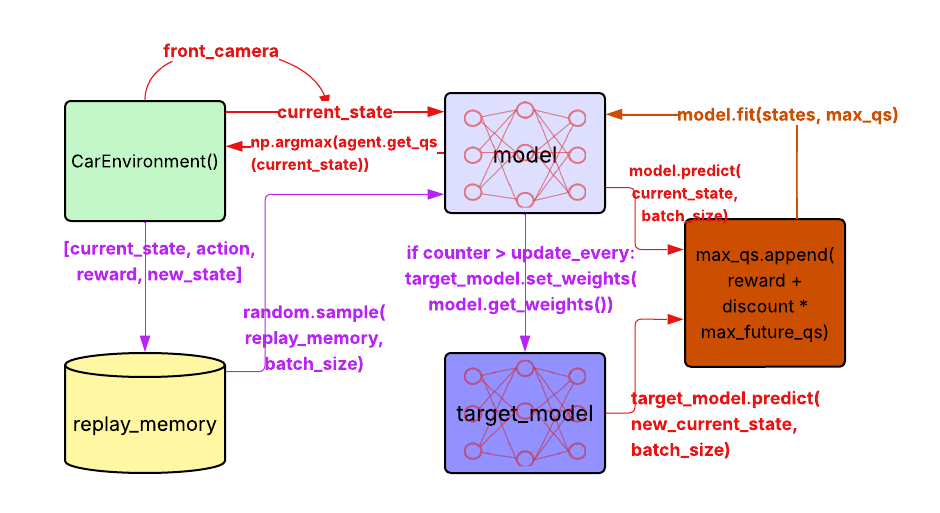
AI-generated content may be incorrect. A video game of cars on a road

AI-generated content may be incorrect.

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.

## Deep Q-Network Implementation

### Initial Setup



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 were 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.

A screen shot of a computer program

AI-generated content may be incorrect.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 Section 3.3, this functionality was limited to the CPU.

### 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 relied on legacy version of TensorFlow and Keras, which enable features such as customised TensorBoard outputs, but were 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.

If you are developing software or hardware, you must test it. This section should explain how your work will be (or has been) tested.

You should have a test plan at the very least (full details of it and its results if required can go in an appendix). Ideally, you will have automated tests for any software you build. You will also define user acceptance tests, or something similar which can be used to determine whether your output meets the requirements stated earlier. Explain how and when the tests should be conducted.

## 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 |

A collection of additional debugging and performance review features were 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.

# External concerns

This section considers factors external to your project. In your PDD, you considered external factors which might influence or be influenced by the process of undertaking your project. In this section, you should consider the way in which your project, its results, any knowledge gained from it or its commercial exploitation might affect the wider world.

## Legal, social and ethical issues

In your PDD, you considered the ethics involved with *doing* your project. Now you should identify and discuss the legal, social, ethical and professional issues raised by the project as a whole, and any artefacts you produced. If you created software to track individuals, or a laser-wielding robot hardware, what effects on the world in general need to be considered? How will you address these issues. Many things can be used for good or bad purposes. Can yours, and what will you do about it?

## Commercial issues and exploitation

Think about what you have created during this project. It may be hardware, software, knowledge or a combination of these. How could you exploit these commercially? Discuss the merits of your work, other commercial entities which might be similar, and consider the costs and potential profits of using your work commercially.

# Evaluation and discussion of results

This section evaluates the *software, hardware or other artefact* you have developed. You should compare it with the original specification and see how well it satisfies the requirements. You may wish to refer back to your aims and objectives at this point. You should report the results of user testing and a summary of feedback if that has been collected.

Evaluation should be rigorous and objective. Avoid opinion and unsubstantiated statements.

If you have done experiments, then the results of these should be reported and discussed here.

If you have involved people in doing user evaluations, that information should be include here.

Overall review of all thing

A screenshot of a computer

AI-generated content may be incorrect. A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.A screen shot of a computer

AI-generated content may be incorrect.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

## Programme specific concerns

Although this does not have to be a section on its own, your report must make clear how the work you have done demonstrates the programme specific competency for your degree programme. There is a full list of these in the assignment specification.

# Conclusion

In this section you should evaluate the *project* as a whole, and the *process* by which you undertook it.

## Project management

What methodology did you actually use? Was it the same as you originally planned? Was your methodology appropriate (and did you stick to it)? Was your time planning good? Did you meet regularly with your supervisor? What changes did you make as a result of your midway review meeting? What have you learned from the process? What would you do better/differently if you had more time? Include the final version of your Gantt chart and comment on its use.

## Risk management

In your PDD you conducted an initial risk management exercise. How appropriate did this turn out to be? If any of the predicted problems occurred, how good were your mitigation plans? If other events happened which caused problems, what could you have done to predict and mitigate them?

## General conclusions

In this section you should evaluate the *project* as a whole, and draw conclusions from the work you have done. Ask yourself what the project has achieved – what is its contribution? Has it met its initial aims and objectives? If not, why? How does the work you have done enhance the field in general? What has been learned from the project? If you have a well defined research question, has it been answered? What do the results mean?

Sometimes, it’s appropriate to include a subsection on ‘Further work’, making suggestions of how to proceed and what could be done to enhance the project in future.

References

References must be formatted in the correct manner. For this assignment you must use the University of Hull’ approved variant of the Harvard referencing style (Fallin 2019), fully described at https://libguides.hull.ac.uk/referencing/harvard. Note that the details of the expected format vary depending on the type of document being referenced. Make sure you are familiar with them. If you use reference management software such as Zotero, EndNote or RefWorks, then you should be able to export a table of references in the correct format, which will save you work.

Every reference should have at least one citation in the text. Most will probably be in the ‘Background’ or ‘Literature review’ sections.

Remember that there is a difference between references and a bibliography. You will certainly need references, but a bibliography is optional.

There is much more information and guidance about referencing on the library’s website at https://libguides.hull.ac.uk/referencing/home

Some examples, illustrating different types of source:

Bahraini, M.S., Bozorg, M., Rad, A.B., (2018). SLAM in dynamic environments via ML-RANSAC. *Mechatronics* 49, 105–118.

Fallin, L., (2019)*. LibGuides: Referencing your work: Harvard Hull.* Available online: http://libguides.hull.ac.uk/referencing/harvard (accessed 10/10/2019).

Janis, I., (1972). *Victims of Groupthink: A psychological study of foreign-policy decisions and fiascoes.* Houghton Mifflin, Boston.

Office For Students (2018) *. Securing student success: Regulation framework for higher education in England*. Available online: https://www.officeforstudents.org.uk/media/1406/ofs2018\_01.pdf (accessed 10/10/2019)

Schmuck, P., Chli, M., (2019). CCM-SLAM: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics* 36, 763–781.

Appendix A – Interesting but not vital material

Appendices are used to include information which may be of interest but is not necessary for the reader. *You do not have to include appendices if there is no need for them*.

You might, for example, want to include some details of a particular piece of software (an API, perhaps) or hardware which your project uses. This might be something that a reader might wish to consult, but you wouldn’t want to include in the main body of the report. You could also put raw data from experiments in an appendix, or perhaps survey results. It should still be information of relevance, but nothing that everyone would be expected to read.

If you wish to refer to elements of your PID, you could include them in appendices.

Appendix B – Other things which may be useful

You can have more than one appendix, or none at all. Give them meaningful names and titles (not the ones given here), so that you can refer to them in the text, and so that they appear in the table of contents.