

43008: Reinforcement Learning

Assignment-3, Part-E Project Report

Project Title:

Car Racing Agent Using Reinforcement Learning

Team Name and Project Number:

F1 Racers

(Project No. 4)

Team Members:

Robayed Ashraf	25088612	Robayed.ashraf@student.uts.edu.au
Asim Santos Poudel	25020835	Asimsantos.poudel@student.uts.edu.au
Chong Fai Wong	24881575	Chong.f.wong@student.uts.edu.au

Date: November 11, 2024

Table of Contents

<i>Abstract.....</i>	<i>3</i>
<i>1. Introduction and Background.....</i>	<i>3</i>
<i>2. Overview of the architecture/system</i>	<i>7</i>
<i>3. Results and Evaluation</i>	<i>11</i>
<i>4. Discussion and Conclusions</i>	<i>16</i>
<i>5. References</i>	<i>17</i>
<i>Appendices</i>	<i>18</i>

Abstract

This project applied reinforcement learning (RL) to train an agent to navigate a simulated racetrack in the Gymnasium Car Racing environment. Three RL algorithms were tested: Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC) to determine their effectiveness in teaching the agent driving strategies like maintaining control, maximizing speed, and staying on the track. The agent's performance was evaluated based on episode rewards, track completion rates, and stability of actions. The results showed differences in each algorithm's approach to decision-making in continuous control tasks, highlighting the potential of RL for autonomous navigation. This project aimed to contribute to understanding RL applications in real-time environments, which could support advancements in autonomous driving technologies.

1. Introduction and Background

This project aimed to explore reinforcement learning (RL) methods to train an agent for autonomous navigation on a simulated racetrack. The task involved creating an agent capable of learning optimal driving behaviours, such as maintaining control, speed, and track adherence through continuous interaction with the environment. In recent years, RL has been increasingly applied to solve complex, sequential decision-making problems, where actions impact future states and long-term rewards. Autonomous navigation, particularly in vehicle control, aligns closely with the goals of RL, as it requires an agent to make real-time, strategic decisions based on ongoing environmental feedback.

The chosen environment for this project, Gymnasium's Car Racing, provided a suitable platform for testing driving tasks in a controlled, simulated setting. By experimenting with three RL algorithms Deep Q-Network (DQN), Proximal Policy Optimization (PPO), and Soft Actor-Critic (SAC), the project aimed to identify effective approaches for continuous control in autonomous navigation. The insights gained from this project may contribute to the growing body of knowledge on RL applications in real-time, dynamic environments, supporting future developments in autonomous driving technologies.

1.1 Problem Statement

This project aims to reduce the human error from traditional means of driving. Manual driving solely relies on human judgement, which is subjected to human error and incorrect decisions. Autonomous driving, on the other hand, is based on advanced algorithms and data-driven decisions. This allows autonomous vehicles detect hazards and predict other road users' actions, resulting in safer driving compared to humans. Autonomous driving passengers can use travel time for work, rest, or entertainment, rather than focusing on the road. This can lead to increased productivity and a more enjoyable travel experience.

1.2 Motivation

This project aims to explore the implementation of autonomous navigation with the use of reinforcement learning. Reinforcement learning requires the agent to explore the environment through understanding which actions yield the best reward and generate an optimal policy of what are the best actions to be taken in a certain state. Reinforcement learning is highly suitable for autonomous driving for the following reasons.

Driving requires the driver to react and make sequential decision task based on the state of the road. This makes reinforcement learning a highly compatible approach for autonomous driving as these algorithms target to maximize the reward, which align well with objectives such as safe driving and route optimization.

Moreover, reinforcement models have the capacity in optimizing multiple objectives at the same time, such as minimizing travel time and safe driving. This is thanks to reinforcement learning's reward structure that allows taking multiple objectives into account and quantifying them into evaluation calculations. This allows the model to balance and prioritize different objectives effectively.

Unlike other alternatives which require implementation of incremental learning, reinforcement learning can automatically improve over time by exploring through real world scenarios, thanks to their exploration, exploitation, and reward structure. This continuous learning loop can enhance the vehicle's capabilities as it encounters new scenarios, ultimately making autonomous systems safer and more efficient.

1.3 Application

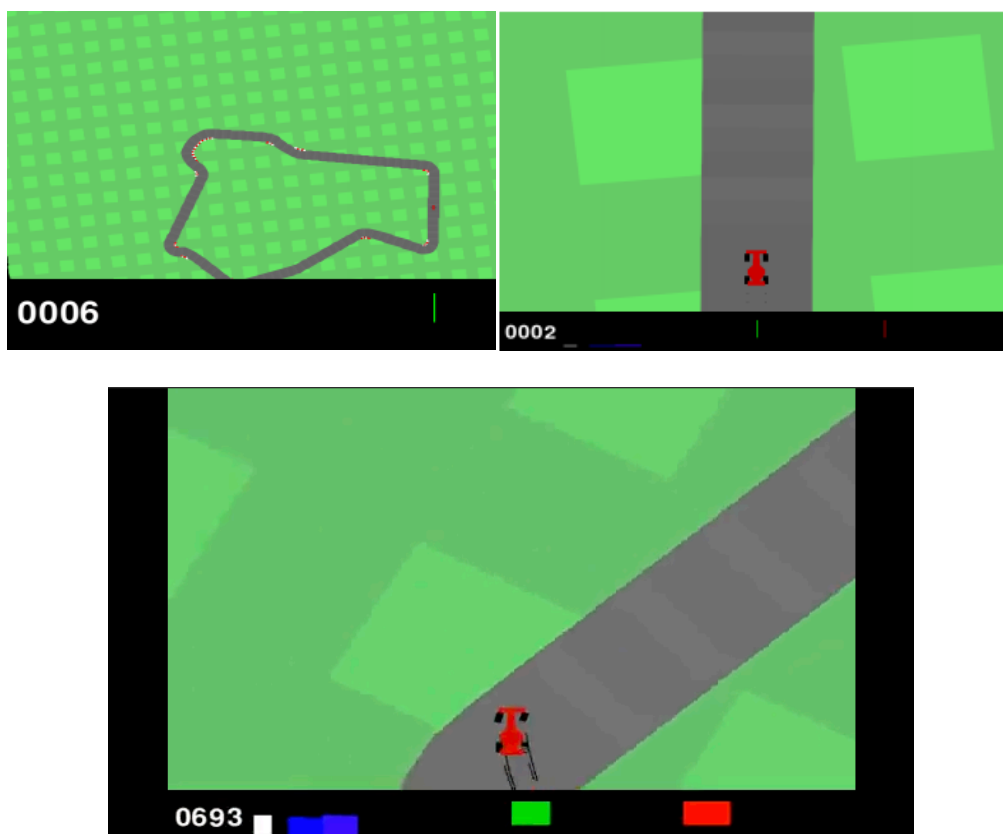
The outcomes of this project have potential applications in the development of autonomous driving systems, where reinforcement learning can enable vehicles to make safe and efficient driving decisions in real time. By learning to navigate a simulated racetrack, the agent gained skills relevant to real-world driving tasks, such as adjusting speed, maintaining track alignment, and optimizing control under varying conditions. Insights from this project could be applied to improve vehicle navigation systems, enabling autonomous cars to react more effectively to their surroundings and adapt their strategies over time. Additionally, this work may support advancements in other fields that require continuous control and sequential decision-making, such as robotics, drone navigation, and intelligent traffic systems.

1.4 Dataset/Environment

The environment used for this project was the Car Racing environment from Gymnasium's Box2D suite. This environment provides a simulated 2D racetrack with a top-down view, allowing the agent to navigate using continuous control inputs. The observation space consists of 96x96 RGB images, showing the car's position on the track and the surrounding area. The action space includes three continuous controls: steering (left or right), gas (accelerate), and brake.

Action Space	Box([-1. 0. 0.], 1.0, (3,), float32)
Observation Space	Box(0, 255, (96, 96, 3), uint8)
import	<code>gymnasium.make("CarRacing-v3")</code>

The environment assigns rewards based on the agent's progress on the track. A small time-penalty (-0.1 points per frame) discourages excessive delays, while a reward is granted for visiting new sections of the track. The episode concludes when the agent completes all track tiles or leaves the track boundaries, which results in a penalty. This setup effectively encourages the agent to drive efficiently and remain on track.



Sample images from the environment illustrate the top-down view of the track and the red car representing the agent. These visuals, along with status bars for acceleration, steering, and braking, provide feedback on the agent's real-time interactions within the environment.

(Klimov, 2014)

1.5 RL Algorithm Comparison

(Stable Baselines, 2018)

Algorithm	DQN	SAC	PPO
Exploration	ϵ -greedy policy	Entropy term in the reward	Stochastic policy
Action Space	Discrete	Continuous	Continuous & Discrete
Off/On Policy	Off-policy	Off-policy	On-policy
Use Cases	Simple, discrete environments (e.g., Atari games)	Complex, high-dimensional continuous action spaces (robotics, control tasks)	General-purpose, complex continuous or discrete tasks
Key Advantage	Simplicity, off-policy nature (sample efficiency)	Handles continuous action spaces well, entropy maximization	Balanced between ease of use and performance

2. Overview of the architecture/system

2.1 Workflow and Flowchart

Workflow Description:

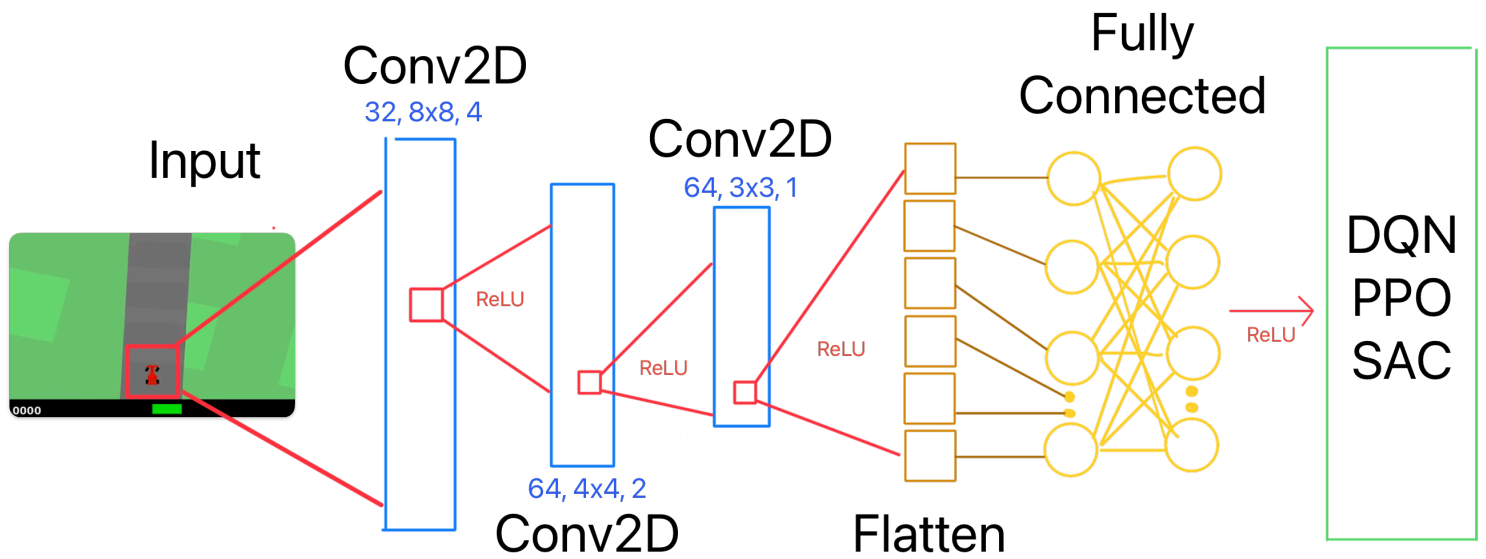
- i. **Environment Initialization:** The Gymnasium Car Racing environment is initialized, setting up the racetrack and the initial position of the agent (car).
- ii. **State Observation:** At each time step, the agent receives a 96x96 RGB image of the racetrack, representing the current state.
 - **Action Selection:** The agent selects an action (steering, acceleration, or brake) based on the current state. The method for action selection depends on the chosen algorithm:
 - **DQN:** Actions are selected based on Q-values.
 - **PPO:** Actions are selected using a policy network that outputs a probability distribution over possible action.
 - **SAC:** Actions are sampled from a stochastic policy to encourage exploration.
- iii. **Experience Storage (DQN & SAC):** For DQN and SAC, each experience (state, action, reward, next state) is stored in a replay buffer for later use in updating the model.
- iv. **Reward Calculation:** The environment provides feedback in the form of rewards. Positive rewards are given for progressing along the track, while penalties discourage time delays and off-track actions.
- v. **Policy Update:** Each algorithm updates its policy based on the received rewards:
 - **DQN:** Updates Q-values using the Bellman Equation.
 - **PPO:** Updates the policy with a clipped objective function to stabilize learning.
 - **SAC:** Updates the actor and critic networks, with entropy maximization to maintain exploration.
- vi. **Termination Check:** At each step, the algorithm checks if the episode has ended. If the episode is complete (either by reaching the end of the track or going off-track), the environment resets for the next episode. If not, the agent observes the next state and continues the loop.
- vii. **End:** The training session ends after completing the designated number of time steps or episodes.

The flowchart below visually represents this workflow, illustrating the paths taken by each algorithm from state observation through action selection, reward calculation, and policy updates. Each step is connected to show the logical flow and sequence of operations for effective training.

Flowchart



2.2 System Architecture Design



The figure above shows the CNN architecture adopted by all three models. The configuration of the three layers is shown below:

Layer	Input	Output	Kernel Size	Stride	Activation
First Convolutional Layer	96 neurons (from observation space)	32 neurons	8x8	4	ReLU
Second Convolutional Layer	32 neurons	64 neurons	4x4	2	ReLU
Third Convolutional Layer	64 neurons	64 neurons	3x3	1	ReLU
Flattened Fully Connected Layer	64 neurons (flattened to 1D)	512-dimensional feature representation	-	-	-

2.3 GUI/Environment Design

The GUI of the Car Racing environment provides a simplified top-down perspective, allowing the agent's interactions and movement along the racetrack to be clearly visualized. This design enables effective evaluation of the agent's actions, rewards, and progress within the environment. It facilitates the assessment of the agent's learning and behavioural adjustments as it navigates the track, responding to reinforcement signals. This setup supports reinforcement learning by providing continuous visual feedback on the agent's performance and adaptations.

GUI Components:



- i. **Environment:**
Displays a top-down view with the grey track representing the racetrack and the green area showing the surrounding landscape.
- ii. **Agent (Red Car):**
The red car represents the agent, which navigates and interacts with the environment.
- iii. **Score Display:**
The numeric value in the bottom left corner indicates the cumulative reward (score) earned by the agent.
- iv. **Progress and Status Bars:**
 - a. **White Bar:**
Tracks the agent's progress by showing the exploration of unique tiles on the racetrack.
 - b. **Blue Bar:**
Indicates the car's acceleration level.
 - c. **Green Bar:**
Reflects the steering direction (left or right).
 - d. **Red Bar:**
Represents the braking intensity applied by the car.

3. Results and Evaluation

3.1 Experimental Settings

The tables below shows the hyperparameter configurations for each model used in this project. The environment that this experiment was conducted was in the Car Racing (Version 3) environment from Gymnasium. Each model utilizes a CNN for policy-based feature extraction, as the observation space consists of 96x96 pixel RGB images from a top-down perspective, making CNNs well-suited for this task.

DQN

Policy	Learning Rate	Buffer Size	Exploration Fraction	Exploration Final Episodes	Train Frequency	Batch Size	Total Timesteps
CNN	0.00025	100000	0.1	0.01	4	256	4.6 million

PPO

Policy	Learning Rate	Clip Range	Ent_coef	n_epochs	n_steps	vf_coef	Batch Size	Total Timesteps
CNN	2.5e-4	0.1	0.01	4	1024	0.5	512	6.1 million

SAC

Policy	Learning Rate	Buffer Size	Ent_coef	Gamma	Grad Steps	Training Frequency	Batch Size	Total Timesteps
CNN	0.0003	10000 0	auto	0.98	32	1024	512	5.2 million

3.2 Pre-Processing

```
from gymnasium import spaces
# Discrete action mapping for CarRacing-v3
action_mapping = [
    np.array([0.0, 0.0, 0.0]), # 0: do nothing
    np.array([-1.0, 0.0, 0.0]), # 1: steer left
    np.array([1.0, 0.0, 0.0]), # 2: steer right
    np.array([0.0, 1.0, 0.0]), # 3: gas
    np.array([0.0, 0.0, 0.8]) # 4: brake
]

# Custom wrapper to convert discrete actions to continuous actions
class DiscretizedCarRacingEnv(gym.Wrapper):
    def __init__(self, env):
        super(DiscretizedCarRacingEnv, self).__init__(env)
        self.action_space = spaces.Discrete(len(action_mapping)) # Override action space

    def step(self, action):
        continuous_action = action_mapping[action] # Map discrete action to continuous
        obs, reward, terminated, truncated, info = self.env.step(continuous_action) # Unpack all five return values
        return obs, reward, terminated, truncated, info # Return exactly 5 values as expected by Gymnasium

    def reset(self, **kwargs):
        return self.env.reset(**kwargs)
```

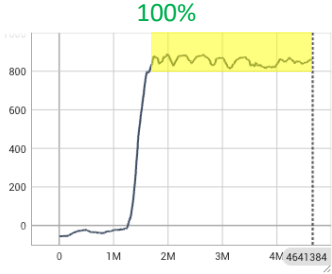
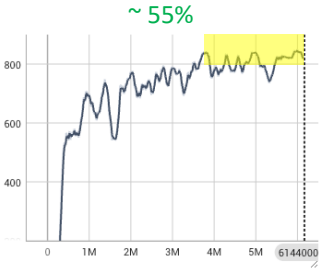
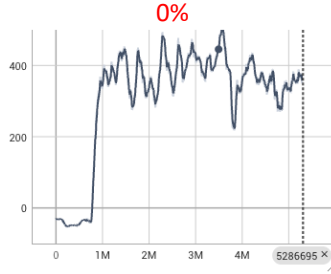
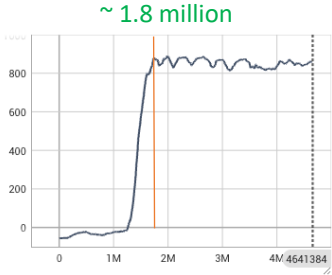
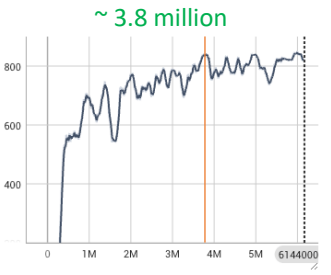
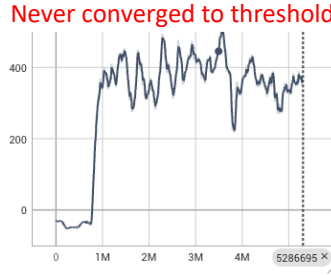

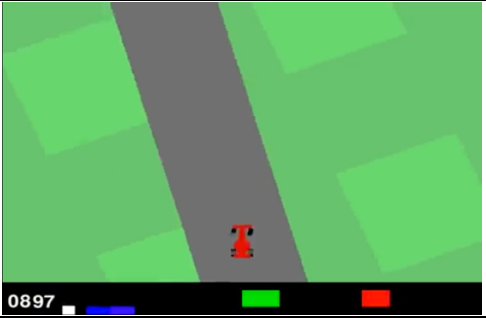

The figure above shows the discretization process when setting up the environment for training DQN. As DQN is designed for discrete action space, while the Gymnasium car racing environment has a continuous action space, the default environment is not compatible with the reinforcement learning model. To make DQN a viable option for training a car racing agent, discretization is implemented. Actions from the environment has been divided into 5 actions: do nothing, steer left, steer right, gas, and brake in the `action_mapping` variable. The variable then replaces the original action space and is being implemented as a customized environment. On the other hand, the environment for PPO and SAC are left untouched since both algorithms are compatible with the environment's continuous action space.

3.3 Experimental Results

Training Outcomes

Algorithm	DQN	PPO	SAC
Timesteps	4.6 million	6.1 million	5.2 million
Training stats	<pre> rollout/ ep_len_mean 931 ep_rew_mean 853 exploration_rate 0.01 time/ episodes 4808 fps 64 time_elapsed 71864 total_timesteps 4637384 train/ learning_rate 0.00025 loss 0.638 n_updates 283586 </pre>	<pre> rollout/ ep_len_mean 980 ep_rew_mean 818 time/ fps 71 iterations 750 time_elapsed 86179 total_timesteps 6144000 train/ approx_kl 0.013364861 clip_fraction 0.315 clip_range 0.1 entropy_loss -5.32 explained_variance 0.986 learning_rate 0.00025 loss 2.62 n_updates 2996 policy_gradient_loss 0.01 std 1.5 value_loss 14.5 </pre>	<pre> rollout/ ep_len_mean 1e+03 ep_rew_mean 360 time/ episodes 5300 fps 75 time_elapsed 70178 total_timesteps 5278695 train/ actor_loss -14.9 critic_loss 3.29 ent_coef 0.00716 ent_coef_loss 0.629 learning_rate 0.0003 n_updates 161824 std 0.0501 </pre>
Episode Length over Time			
Episode Reward Over Time			

Performance Evaluations

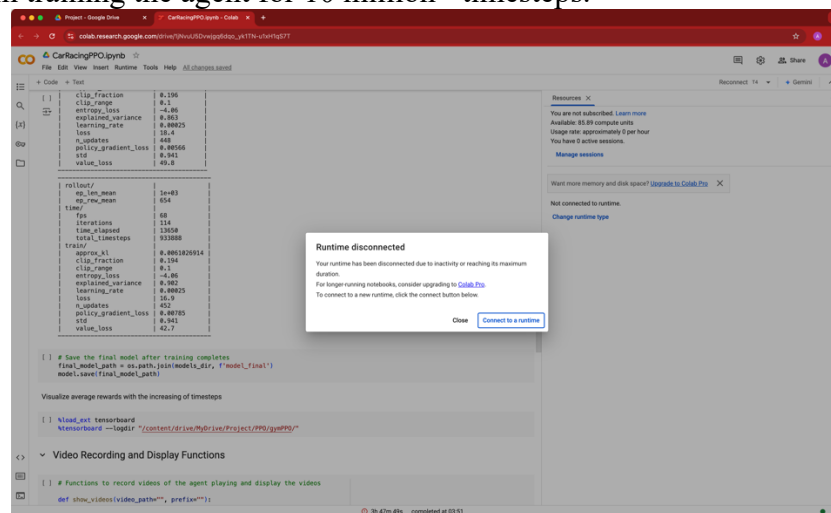
Algorithm	DQN	PPO	SAC
Score Threshold	800 points	800 points	800 points
Success Rate (after convergence)	 <p>100%</p>	 <p>~ 55%</p>	 <p>0%</p>
Average Reward	853	818	360
Average ep Length	931	980	1e+03
Loss	0.638	2.62	Actor (-14.9) Critic (3.29)
Convergence Timestep (Based on observing the rewards graphs)	 <p>~ 1.8 million</p>	 <p>~ 3.8 million</p>	 <p>Never converged to threshold</p>
Training Time	~ 20 hours	~ 24 hours	~ 20 hours
Outcome (Screenshot from Video)			

3.4 Limitations

The project during its implementation, encountered a lot of limitations and challenges. These limitations could be grouped into two categories: Computing Resources specific and Reinforcement Learning specific.

Computing Resources Constraints:

- Reinforcement Learning demands high computational capabilities. Since our programming environment was primarily in Google Colab, the free version was inadequate for the computational demands of the project. So, we had to upgrade to the Pro version and buy additional computing units. Over the project duration, this incurred huge expenses for the group.
- The Colab notebooks got frequently disconnected from runtime after running a maximum of 20 hours, which caused interruptions in the model training and limited the group from training the agent for 10 million+ timesteps.



Reinforcement Learning Constraints:

- Reinforcement Learning requires prolonged timesteps during training to observe if an agent is learning properly. This long wait between training and assessing, made the process of debugging very time-consuming. Each cycle of hyper-parameter tuning and testing took several days which slowed down overall progress.
- The basic CNN architecture was chosen to reduce training time. If the above-mentioned limitations didn't exist, we could've experimented with more sophisticated architectures which potentially could've provided better feature extractions and helped the agent to learn better.

4. Discussion and Conclusions

It is shown in the results that DQN and PPO successfully reaches convergence at a score of around 800, while SAC is not able to converge. DQN is shown to have the best results in terms of success rate, average reward, and loss. It is also the most computationally efficient model among the three. DQN converges at around 2 million timesteps with an average reward of 853 timesteps while PPO require 24 hours to converge around 4 million timesteps with an average reward of 818, and SAC never reaches convergence with an average reward of 360.

The better results of DQN when comparing to SAC and PPO can be attributed to the following reasons. DQN is a RL technique developed by combining value-based Q learning with neural network approximation of Q values. Q learning is one of the most efficient RL models as it only aims to obtain the optimal policy by estimating the maximum discrete Q value of the next state. Since the continuous environment is discretized for compatibility with DQN, this makes training DQN faster than SAC and PPO because Q-learning updates are based on maximizing over discrete action values rather than approximating and sampling a policy distribution.

DQN is able converge faster than PPO and SAC are due to discretized action space and its reduced complexity compared to PPO and SAC. The action space of the car racing environment is being discretized during DQN training. This means that the agent only needs to select from a fixed number of actions. On the other hand, the action space for SAC and PPO are kept continuous. This allows DQN to converge faster since it learns algorithm does not need to optimize a continuous action distribution (as in SAC or PPO). Moreover, PPO and SAC are more complexed than DQN since they are policy gradient methods that require extra mechanisms for entropy regularization. For example, PPO adopts clipping and SAC samples from continuous distributions. These additional parameters can lead to computational overhead, and slowdown the training process. The clipping mechanism of PPO also leads to slower updates in exchange for smooth convergence, requiring more computational timesteps to reach convergence. On the other hand, DQN has a simpler structure than both PPO and DQN. This could explain why DQN trains faster and reaches optimal results at a shorter time than DQN and PPO.

The results above shows that our SAC model never reaches convergence given 24 hours of training. The following could be possible implications. Firstly, the entropy coefficient might be suboptimal. SAC uses an entropy term to encourage exploration. The entropy coefficient controls how much exploration is encouraged. According to the results the average reward for SAC starts to throttle and fluctuate around 400. This could imply that exploration is not encouraged enough, leading to the model settling in a suboptimal policy, even SAC has automatic entropy tuning. The results could also be attributed to poorly tuned learning rates. Training SAC requires tuning of actor and critic learning rate simultaneously. This means that if either one of them are too high or too low, it could lead to value functions oscillating or a slow convergence respectively. In our case, it is indicated that critic loss is way higher than the actor loss, in which the learning rate of critic loss being suboptimal.

One major improvement that could be made is to speed up the training process of PPO and SAC. Hyperparameter tuning, including a large learning rate, larger learning rate for critic, normalizing the rewards to improve the performance of PPO and SAC. Self-attention layers could be added to the custom CNN network to help the agent focus on important parts of the state space. To facilitate a more efficient exploration process, action loss, such as Ornstein-

Uhlenbeck, can be implemented to encourage exploration such as local optima can be broken through. Intrinsic rewards could also be applied to encourage the agent to explore unknown states in car racing. Moreover, our team would also explore the use of more advanced RL algorithms such as Double DQN to reduce overestimation of Q-values in DQN, which could lead to suboptimal policies as the agent may incorrectly predict that certain actions will yield higher rewards than they do.

5. References

Klimov, O. (2014). Gymnasium Documentation. Retrieved from The Farama Foundation: https://gymnasium.farama.org/environments/box2d/car_racing/

Stable Baselines. (2018). Stable Baselines Docs. Retrieved from Stable Baselines: <https://stable-baselines.readthedocs.io/en/master/guide/algos.html>

Appendices

A. Individual Contribution

The problem statement, motivation, application, and environment/dataset descriptions were among the components of the report that I was mostly responsible for researching and writing. I thoroughly looked at the difficulties of applying reinforcement learning (RL) to autonomous driving, concentrating on how RL might assist a car in making safer and more effective judgments. This effort was crucial for establishing precise project objectives and outlining the rationale behind our strategy.

I also set up the initial hyperparameters for the DQN algorithm. This required selecting parameters that would allow quicker and more reliable learning, enabling the model to get better as it trained. I installed the required libraries and dependencies and made sure the dataset was available when I set up the training environment in Google Colab. To execute many algorithms smoothly and prevent technical problems during training, this setup procedure was crucial.

I detailed the experimental setup and the preliminary results from training each algorithm in the Results and Evaluation section. Here, I was responsible for outlining the training settings and documenting the first performance of each algorithm. To illustrate our findings, I made several charts that displayed the evaluation metrics and training progress, which enabled us to assess the advantages and disadvantages of each method. After completing the initial task, I had to tune my hyperparameter for DQN several times to get the best model out of it. Then, I kept the best configuration for final training, and the agent started learning fast and achieved the benchmark so fast.

Lastly, I reviewed the complete report draft to make sure it was coherent, consistent, and well-structured. This involved ensuring that each component was accurate, that the material matched the objectives of our project, and that the reader could understand the language.

B. Individual Contribution Split

Team Member Name	Percentage
Robayed Ashraf	33%
Asim Santos Poudel	33%
Chong Fai Wong	33%
Lecturer and Tutor	1%

Robayed Ashraf: *robayedashraf*

Asim Santos Poudel: *asimsantos*

Chong Fai Wong: *chongfaiwong*