# Performance Comparison of Neural Networks and Traditional Regression Models on Diverse Datasets

**David Bishop** DBISHOP7@JH.EDU
*Computer Scientist*
*John's Hopkins University*
*Washington, DC 20001, USA*

**Editor:** David Bishop

## Abstract

This paper investigates the performance of three reinforcement learning algorithms—Value Iteration, Q-Learning, and SARSA—on a racetrack problem designed to test their ability to navigate through different tracks, sometimes under different crash conditions. Our hypothesis was that Value Iteration, with its model-based approach, would outperform the model-free algorithms (Q-Learning and SARSA) in minimizing the number of steps required to complete the racetrack, especially when harsh crash conditions are applied. The results of our experiments confirm this hypothesis, showing that Value Iteration consistently required fewer steps to reach the finish line, leveraging its comprehensive planning capabilities. Q-Learning and SARSA, while effective, exhibited higher step counts due to their reliance on direct interaction with the environment, which made them more susceptible to the challenges posed by non-determinism and crash penalties. These findings highlight the importance of algorithm selection based on the specific nature of the problem, particularly in environments where long-term planning is essential.

**Keywords:** Reinforcement Learning, Value Iteration, Q-Learning, SARSA, Racetrack Problem, Non-Determinism, Crash Penalty

## 1 Introduction

Reinforcement learning (RL) has become a pivotal approach in solving sequential decision-making problems where an agent must learn to make optimal decisions through trial and error. Among the most commonly studied RL problems is the racetrack problem, a control problem where the goal is to navigate a car from a starting point to a finish line on a predefined track in the shortest possible time (3). The challenge lies in the car's dynamics, the non-deterministic nature of actions, and the penalties associated with crashing into track boundaries.

This paper investigates the performance of three prominent RL algorithms—Value Iteration, Q-Learning, and SARSA—on various racetrack configurations. Value Iteration, a model-based algorithm, utilizes a complete understanding of the environment to iteratively compute the optimal policy by evaluating the expected utility of states (2). In contrast, Q-Learning and SARSA are model-free algorithms that rely on direct interactions with the environment to learn optimal policies. Q-Learning, being an off-policy method, estimates the value of the optimal policy independently of the agent's actions, while SARSA, an on-policy method, learns the value of the policy actually being followed (1).

Our primary hypothesis states that Value Iteration will outperform Q-Learning and SARSA in minimizing the number of steps required to complete a racetrack, particularly under harsher crash conditions where the vehicle is reset to the starting position after a crash. The rationale behind this hypothesis is that Value Iteration, with its complete model of the environment, is better equipped to plan for long-term rewards, whereas Q-Learning and SARSA, due to their model-free nature, might struggle more with the non-determinism and penalties associated with crashes.

To test this hypothesis, we applied these algorithms to four racetrack configurations of varying complexity, including an analysis of their performance under different crash conditions. The results of these experiments provide insights into the strengths and weaknesses of each algorithm in navigating the racetracks efficiently and offer a deeper understanding of how different RL approaches handle the challenges posed by uncertain environments.

## 2 Algorithms and Experimental Methods

In this project, we employed three distinct reinforcement learning algorithms: Value Iteration, Q-Learning, and SARSA. Each of these algorithms have a unique approach to learning optimal policies for decision-making tasks.

### 2.1 Experimental Approach

The datasets used in this project were simple ASCII representations of racetracks, which the models attempted to navigate. Due to the nature of the datasets, no pre-processing steps were required beyond reading the track data into a Python structure. However, code was needed to create a 'Track' class that managed the car's position on the track and detected crashes, as well as a 'Car' class to ensure accurate simulation of the car's movements.

A key aspect of this experiment was introducing randomness in the success of actions. Each time the model decided on an action—such as attempting to accelerate the car up, down, left, or right—there was an 80% chance that the action would fail, simulating real-world uncertainty.

Another critical component was the reward system, which provided feedback to the model on the quality of its decisions. The reward values were designed to reflect the model's performance in different scenarios, with the goal of encouraging efficient and effective navigation of the tracks. The reward structure used for all models was as follows:
- If the model crashed: -100 points.
- If the model crashed in a scenario where it would restart from the initial position: -1000 points. (Note: This was removed for SARSA's run of R-Track resets as adding it caused it to never be able to learn)
- If the model chose not to move during a turn: -100 points.
- If the model remained in the starting positions without advancing: -10 points.
- If the model revisited a previously visited state: -10 points.
- If the model reached the finish line: +100 points.
- For every move the model made: -10 points.

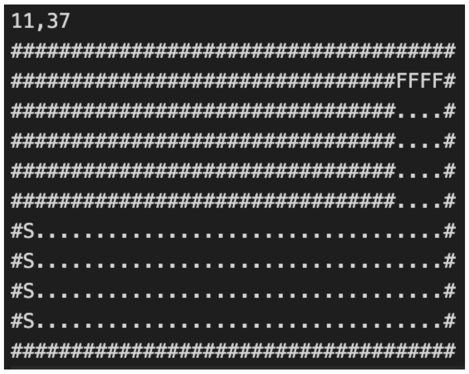
These reward values were determined based on the specific challenges encountered by the models during training and testing. The cumulative reward was calculated and fed back to the model after each action to guide its learning process.

Several hyperparameters needed to be carefully tuned for each algorithm to optimize performance:
- **Value Iteration** required tuning the discount factor ($\gamma$) and the convergence threshold ($\theta$). The discount factor $\gamma$ controls the importance of future rewards compared to immediate rewards. A value of $\gamma = 0.8$ was chosen after experimentation, balancing the model's focus on short-term and long-term rewards. The convergence threshold $\theta$ determines the level of precision needed for the value function; a value of $\theta = 0.001$ was selected as it provided a good balance between computation time and accuracy.
- **Q-Learning** and **SARSA** both required tuning the learning rate ($\alpha$), the discount factor ($\gamma$), and the exploration rate ($\epsilon$).
- The learning rate $\alpha$ controls how much new information overrides old information in the learning process. An $\alpha = 0.1$ was found to be effective, allowing the model to learn steadily without overwhelming previously learned policies.
- The discount factor $\gamma = 0.95$ was chosen to emphasize long-term rewards more heavily than in Value Iteration, as Q-Learning and SARSA typically benefit from considering future rewards more strongly.
- The exploration rate $\epsilon = 0.1$ determines how often the model tries new, unexplored actions versus exploiting known actions. A low $\epsilon$ encourages exploitation of learned policies while still allowing some exploration to discover potentially better strategies.
- The final variable that needed to be set was the number of "episodes" the model would run. This determined how many times the model would iterate through the scenario. Through experimentation, 10,000 episodes were found to be a suitable choice, as they allowed the models to learn the tracks effectively without requiring excessive computation time. These hyperparameters were determined through extensive trial and error, aiming to strike a balance between the exploration of the environment and the exploitation of known strategies, ensuring the models effectively learned to navigate the tracks under the given conditions.

## 2.2 Datasets

For this experiment, we utilized four datasets corresponding to different track configurations. Each dataset was tested under the condition that if a crash occurs, the car is repositioned to the nearest valid position. Additionally, for one track, we tested the model's ability to learn when the crash condition resets the car to its initial starting position instead of placing it at the nearest location. Each track is represented using ASCII characters: 'S' represents the starting positions, 'F' represents the finish positions, '.' represents the track itself, and '#' represents out-of-bounds areas, including walls.

### 2.2.1 L-track.txt

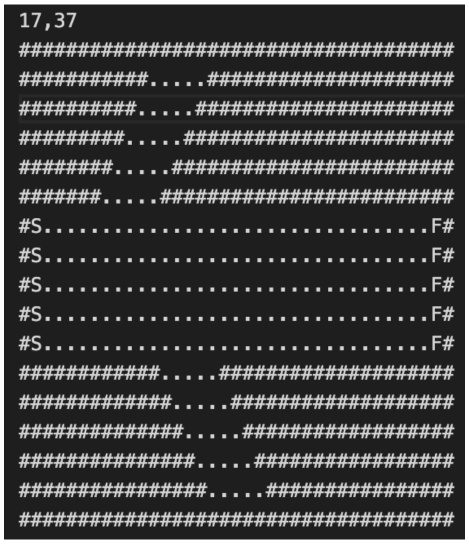


```
11,37
#####################################
################################FFFF#
################################....#
################################....#
################################....#
################################....#
#S..................................#
#S..................................#
#S..................................#
#S..................................#
#####################################
```

Figure 1: L-track.txt

The L-track is one of the simpler tracks due to its straightforward shape. As shown above, the fastest route involves moving right until reaching the wall, and then moving upward until reaching the finish. This course lacks side tracks, minimizing the chances for the model to get lost.

### 2.2.2 W-track.txt

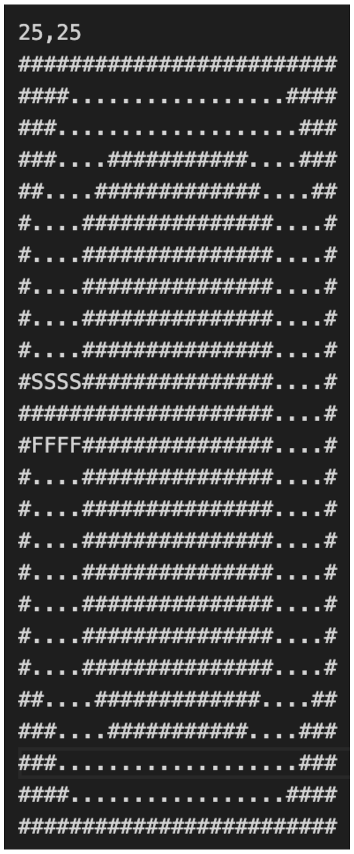


```
17,37
#####################################
###########.....#####################
##########.....######################
#########.....#######################
########.....########################
#######.....#########################
#S.................................F#
#S.................................F#
#S.................................F#
#S.................................F#
#S.................................F#
############.....####################
#############.....###################
##############.....##################
###############.....#################
################.....################
#####################################
```

Figure 2: W-track.txt

The W-track is almost as simple as the L-track, with a key difference: it includes additional offshoots. While the track initially appears to be a straightforward path from start to finish, the presence of these offshoots can mislead the model, causing it to get stuck or lost if it doesn't learn to navigate correctly.

### 2.2.3 O-track.txt

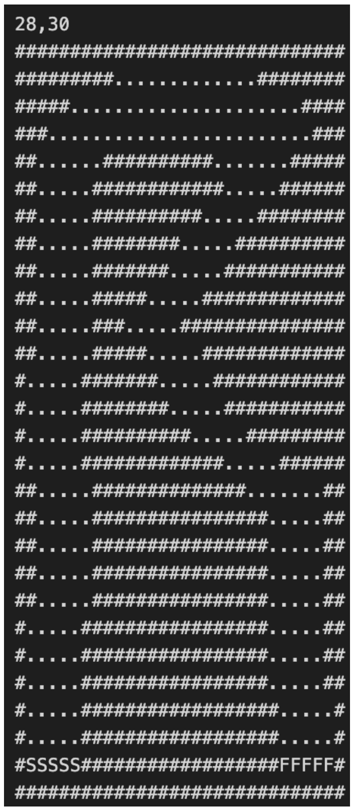


```
25,25
#########################
####.................####
###...................###
###....###########....###
##....#############....##
#....###############....#
#....###############....#
#....###############....#
#....###############....#
#....###############....#
#SSSS###############....#
####################....#
#FFFF###############....#
#....###############....#
#....###############....#
#....###############....#
#....###############....#
#....###############....#
#....###############....#
#....###############....#
##....#############....##
###....###########....###
###...................###
####.................####
#########################
```

Figure 3: O-track.txt

The O-track introduces several features that increase its difficulty for the models. The first challenge is the presence of curves, which require the model to either slow down sufficiently to navigate the turns or crash and then adjust its direction. Another significant challenge is the proximity of the start and finish lines, separated only by a narrow barrier. While a human would recognize that crossing this barrier is not allowed, an inadequately trained model might repeatedly attempt to cross it, believing it could reach the finish.

### 2.2.4 R-track.txt



```
28,30
##############################
#########.............########
#####.....................####
###........................###
##......##########.......#####
##.....############.....######
##.....##########.....########
##.....########.....##########
##.....#######.....###########
##.....#####.....#############
##.....###.....###############
##.....#####.....#############
#.....#######.....############
#.....########.....###########
#.....##########.....#########
#.....#############.....######
##.....##############.......##
##.....################.....##
##.....################.....##
##.....################.....##
##.....################.....##
#.....#################.....##
#.....#################.....##
#.....#################.....##
#.....##################.....#
#.....##################.....#
#SSSSS##################FFFFF#
##############################
```

Figure 4: R-track.txt

The R-track adds further complexity, though not just in its layout. While it shares similarities with the O-track, featuring steep turns, the added challenge comes from the two different crash conditions tested on this track. The first condition, like the others, places the car back on the track at the nearest valid position. The second condition is more stringent, resetting the car to its original starting position after a crash. This increased difficulty forces the models to learn to avoid crashes more effectively.

## 3 Results

In this section, we present the results of each model's performance on each track. Each model was run ten times on each track, and the total number of steps were recorded. By aggregating the results across multiple runs, we ensure that the statistics accurately reflect each model's performance, minimizing the effects of variability in training conditions or other factors that might influence the learning process.

### 3.1 L-Track

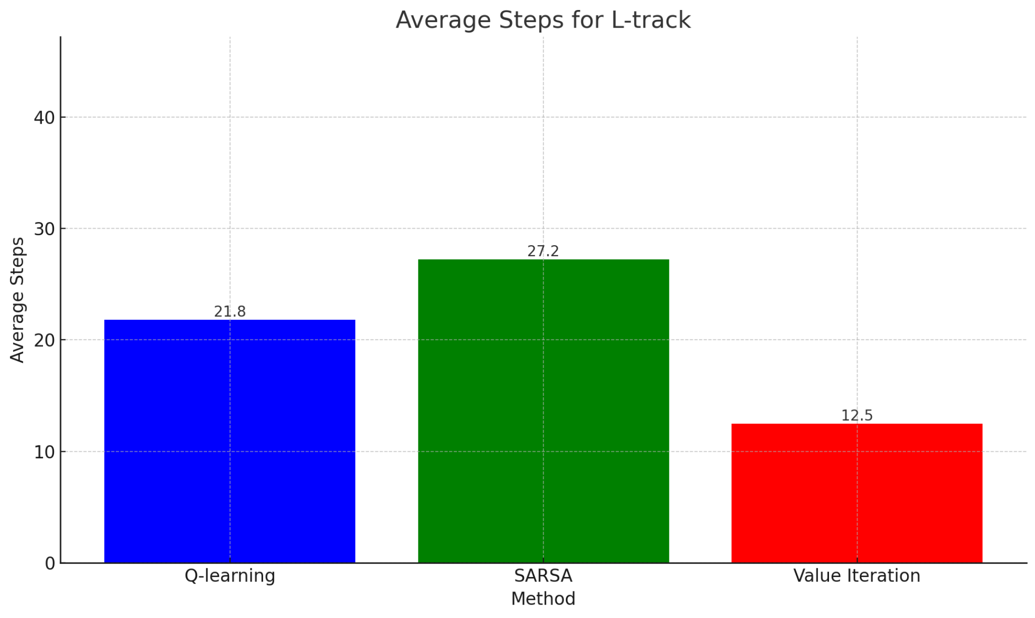


Figure 5: L-track Average Steps

### 3.2 W-Track

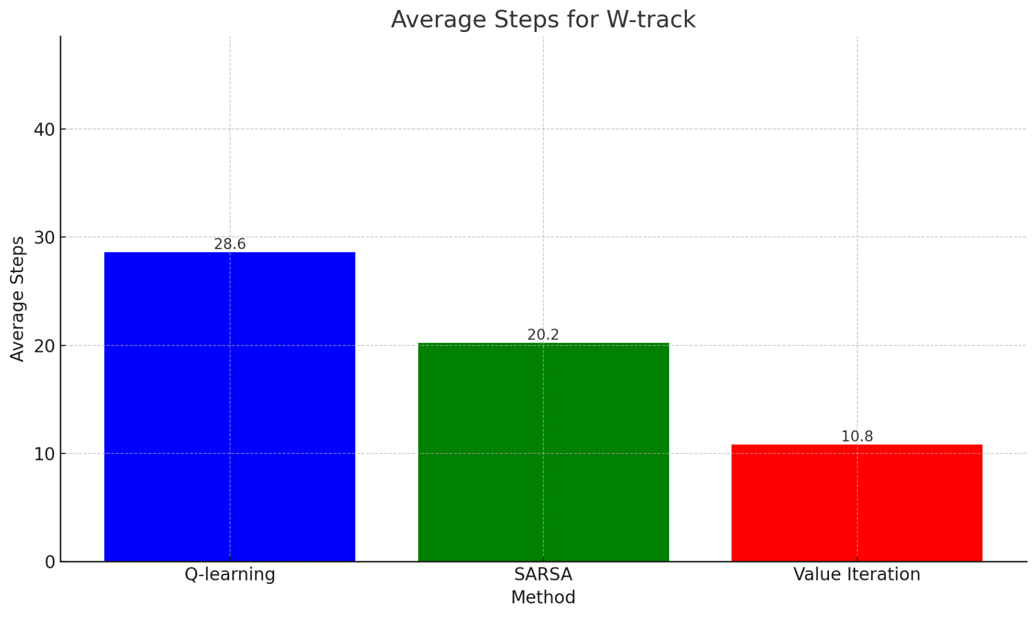


Figure 6: W-track Average Steps

### 3.3 O-Track

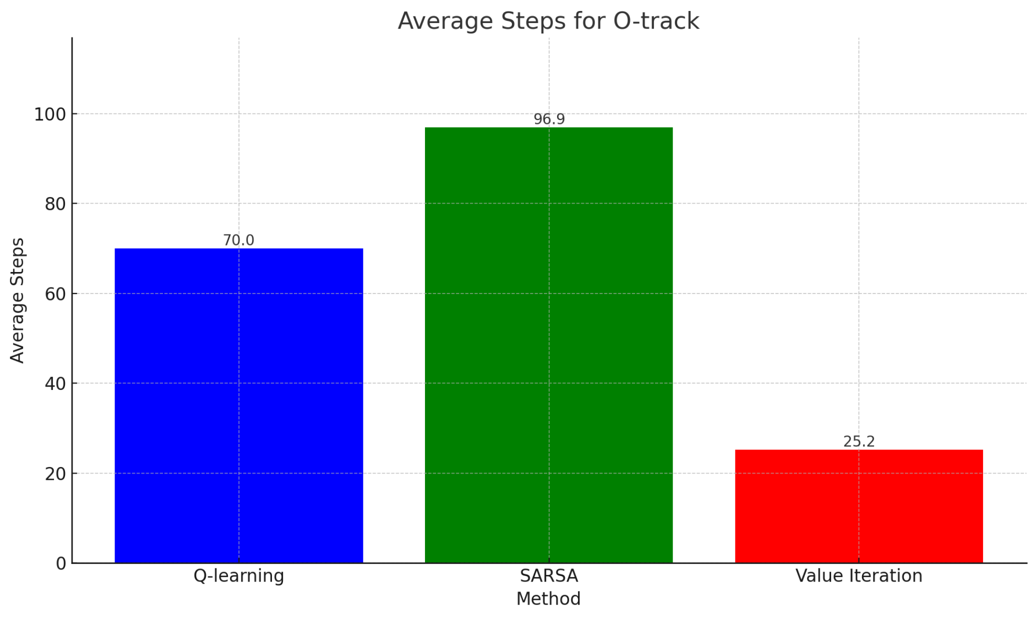


Figure 7: O-track Average Steps

### 3.4 R-Track

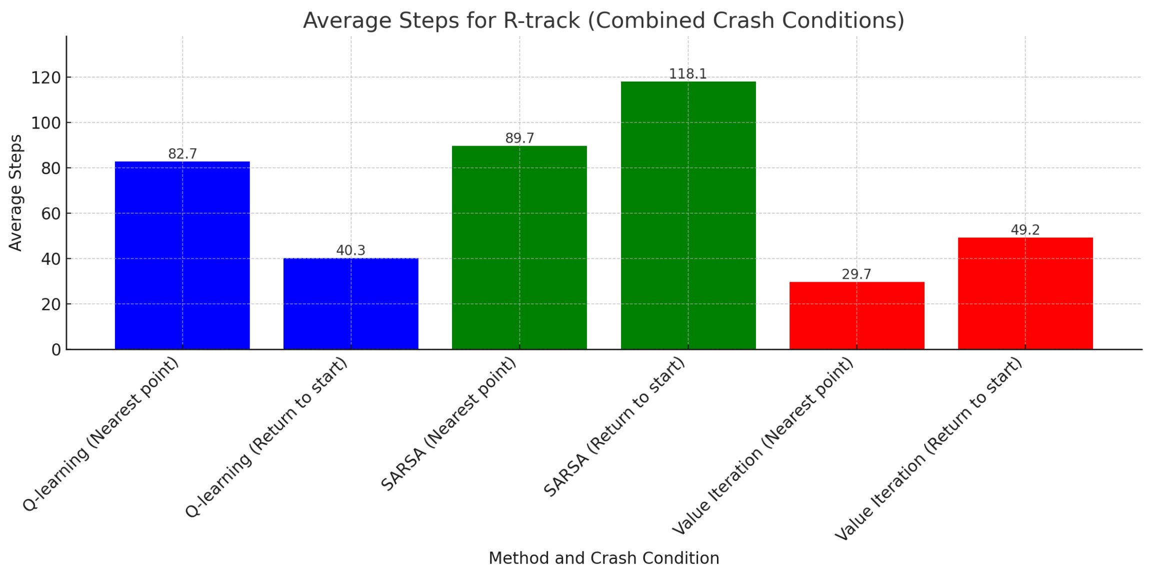


Figure 8: R-track Average Steps

## 4 Discussion

The previous section presented the performance of the models on each of the tracks. The performance results of Q-Learning, SARSA, and Value Iteration across the various racetracks provide valuable insights into the strengths and limitations of these reinforcement learning algorithms. This section goes into the reasons behind these performance outcomes, highlighting how the characteristics of each algorithm align with the specific challenges posed by the tracks.

### 4.1 L-Track

As seen in the graph above L-Track, Value Iteration excelled with an average of 12.5 steps due to its deterministic approach, which quickly identified the optimal path. Q-Learning, averaging 21.8 steps, was slightly less efficient as its off-policy nature led to some unnecessary exploration. SARSA, performing the worst with 27.2 steps, likely struggled due to its on-policy nature, which reinforced suboptimal paths early on.

### 4.2 W-Track

The W-Track's offshoots challenged the models, yet Value Iteration still led with 10.8 steps, efficiently avoiding dead ends through thorough evaluation. Interestingly, SARSA outperformed Q-Learning (20.2 vs. 28.6 steps), likely because Q-Learning's exploration caused it to take suboptimal paths more frequently, while SARSA's on-policy learning found a better path quicker.

### 4.3 O-Track

O-Track's complexity with curves and close start-finish proximity widened the performance gap. Value Iteration completed it in 25.2 steps, leveraging its systematic planning. Q-Learning, needing 70 steps, struggled with the track's demands, while SARSA fared the worst with 96.9 steps, likely due to its reluctance to explore beyond initial poor strategies.

### 4.4 R-Track

There are two Separate parts of R-track, the Nearest Point crash condition and the Return to start crash condition.

#### 4.4.1 Nearest Point

Value Iteration performed well with an average of 29.7 steps, efficiently minimizing crashes. Q-Learning averaged 82.7 steps, as its exploration led to more frequent crashes, though it still outperformed SARSA, which averaged 89.7 steps. SARSA's on-policy approach likely caused it to reinforce less effective strategies, resulting in a higher step count.

#### 4.4.2 Restart

The Return to Start condition imposed a harsher penalty, leading Q-Learning to slightly outperform Value Iteration, with 40.3 steps compared to 49.2. Q-Learning's exploratory nature helped it find safer paths, which was advantageous under this stringent condition.

SARSA struggled the most, averaging 118.1 steps, this is likely due to its inability to handle the new negative reward given to the other models.

## 5 Conclusion

The findings of this study confirm our hypothesis that Value Iteration outperforms Q-Learning and SARSA in minimizing the number of steps required to complete a racetrack, particularly under harsh crash conditions where the vehicle is reset to the starting position after a crash. Value Iteration's model-based approach allowed it to effectively plan for long-term rewards and navigate the tracks more efficiently, demonstrating significant advantages on both simple and complex tracks.

Q-Learning, while generally effective, showed a tendency to struggle with the non-determinism and penalties associated with crashes, particularly in more complex environments. Its off-policy nature sometimes led to unnecessary exploration, which increased the number of steps required to reach the finish line. SARSA, being on-policy, was more conservative but also slower to converge, particularly under harsh conditions, which negatively impacted its performance.

The results underscore the importance of choosing the right reinforcement learning algorithm based on the specific characteristics of the problem at hand. While Value Iteration excels in environments where a complete model of the environment is available and long-term planning is critical, Q-Learning and SARSA offer robust alternatives when such a model is not accessible, albeit with some trade-offs in performance.

Future work could explore hybrid approaches that combine the strengths of these algorithms or adapt them to better handle the non-determinism inherent in real-world applications. Additionally, extending the experiments to include more diverse and dynamic environments could provide further insights into the generalizability of these findings.

## References

[1] Anas, H., Ong, W.H., Malik, O.A. (2022). Comparison of Deep Q-Learning, Q-Learning and SARSA Reinforced Learning for Robot Local Navigation. In: Kim, J., et al. Robot Intelligence Technology and Applications 6. RiTA 2021. Lecture Notes in Networks and Systems, vol 429. Springer, Cham. https://doi.org/10.1007/978-3-030-97672-9_40

[2] Chatterjee, K., Henzinger, T.A. (2008). Value Iteration. In: Grumberg, O., Veith, H. (eds) 25 Years of Model Checking. Lecture Notes in Computer Science, vol 5000. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-540-69850-0_7

[3] Gros, T.P., Höller, D., Hoffmann, J., Wolf, V. (2020). Tracking the Race Between Deep Reinforcement Learning and Imitation Learning. In: Gribaudo, M., Jansen, D.N., Remke, A. (eds) Quantitative Evaluation of Systems. QEST 2020. Lecture Notes in Computer Science(), vol 12289. Springer, Cham https://doi.org/10.1007/978-3-030-59854-9_2