Report

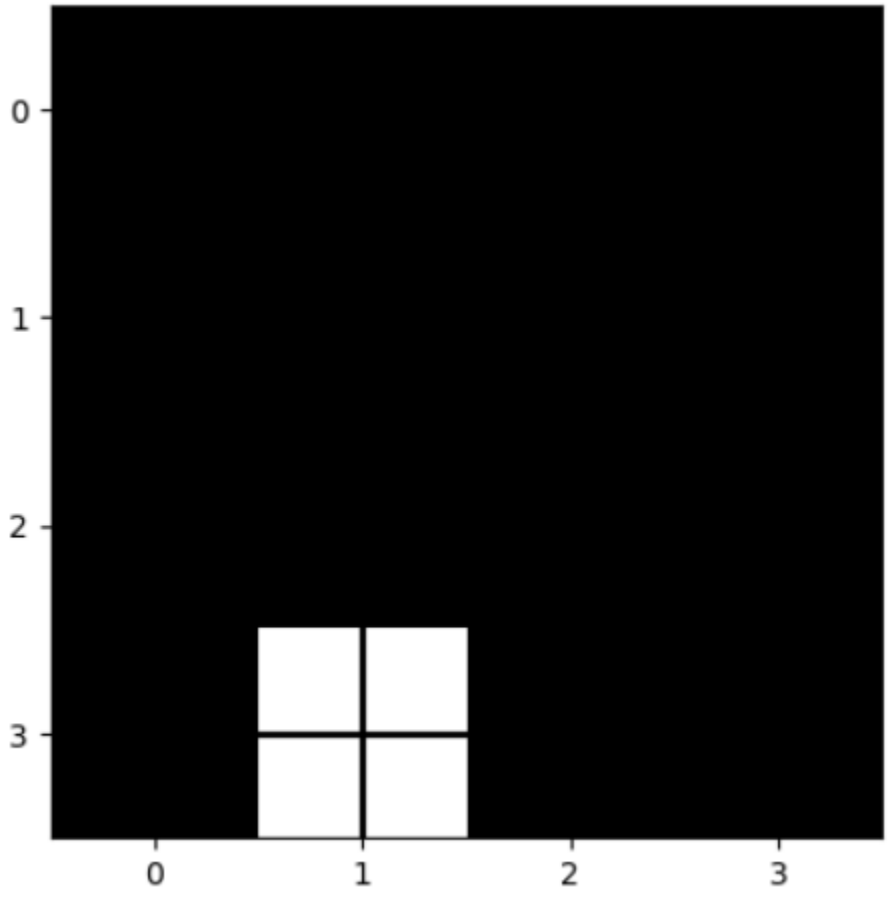
1)Our environment is for mars_exploration_rover whose main objective is to explore the environment while there is a reward for the agent to explore the environment there is also a penalty for moving and a small penalty for exploring as well

The agent has 4 possible actions up, down, right or left to explore the environment

There is a wide range of possible rewards that our model may get
(-exploration_penalty, exploration_reward) gives us the reward range

We have used 16 states for the agent to be in

2) A random presence of our agent in the environment can be given as



3) To ensure the safety of the environment several measures were taken

- Action validation:- In the step method we check if the proposed action is valid by verifying if it leads to a valid state transition

```
def step(self, action):
    reward = 0
    done = False
    i, j = self.state

    if action == 0:
        if i > 0:
            i -= 1
    elif action == 1:
        if i < self.n - 1:
            i += 1
    elif action == 2:
        if j < self.n - 1:
            j += 1
    elif action == 3:
        if j > 0:
            j -= 1
```

- Reward range:-The reward range is set to (-exploration_penalty, exploration_reward), where exploration_penalty is the penalty for stepping on terrain locations, and exploration_reward is the reward for reaching the battery location. This ensures that the rewards returned to the agent fall within this range, promoting safe and stable behavior.

```
self.reward_range = (-exploration_penalty, exploration_reward)
```

Part 2 and Part 3

1)SARSA and Q-learning, two reinforcement learning algorithms for calculating the ideal action-value function, are the primary approaches we have used in this part.

1. SARSA (State-Action-Reward-State-Action):
   Update function: Based on the observed rewards, the current and subsequent states, and the actions, SARSA changes the Q-values.

   The update equation:
   Q(s, a) = Q(s, a) + alpha * (reward + gamma * Q(s', a') - Q(s, a))

   Here,
   - Q(s, a) denotes the Q-value of the state-action pair (s, a)
   - alpha is the rate of learning

- reward denotes the immediate reward
- gamma denotes the discount factor
- s' denotes the subsequent state
- a' denotes the subsequent action.

Key features:-

- SARSA is a policy-following algorithm, which means it adjusts Q-values as it goes along by learning new ones.
- It eventually reaches the best action-value function for the specified policy.

Advantages: SARSA is comparatively easy to use and comprehend.
The best course of action for epsilon-greedy exploration will inevitably be reached.
Environments having a small number of states and actions are ideal for SARSA.

Disadvantages: Because SARSA is an on-policy algorithm, it could not be as efficient with samples as off-policy algorithms like Q-learning.
The selection of hyperparameters, including learning rate and exploration rate, may have an impact on it.

2. Q-Learning:
   Update function:
   Q(s, a) = Q(s, a) + alpha * (reward + gamma * max[Q(s', a')] - Q(s, a))
   Here,
   - Q(s, a) is the Q-value of state-action pair (s, a)
   - alpha is the learning rate
   - reward is the immediate reward
   - gamma is the discount factor
   - s' is the next state
   - a' is the action that maximizes the Q-value in the next state.

   Key features:
   - It uses an epsilon-greedy policy for exploration, similar to SARSA.
   - Regardless of the exploration policy, it eventually converges to the ideal action-value function.

   Advantages:
   - It is capable of navigating settings with a variety of states and activities.
   - Theoretical assurances of convergence to the optimal policy are a well-known feature of Q-learning.

Disadvantages:

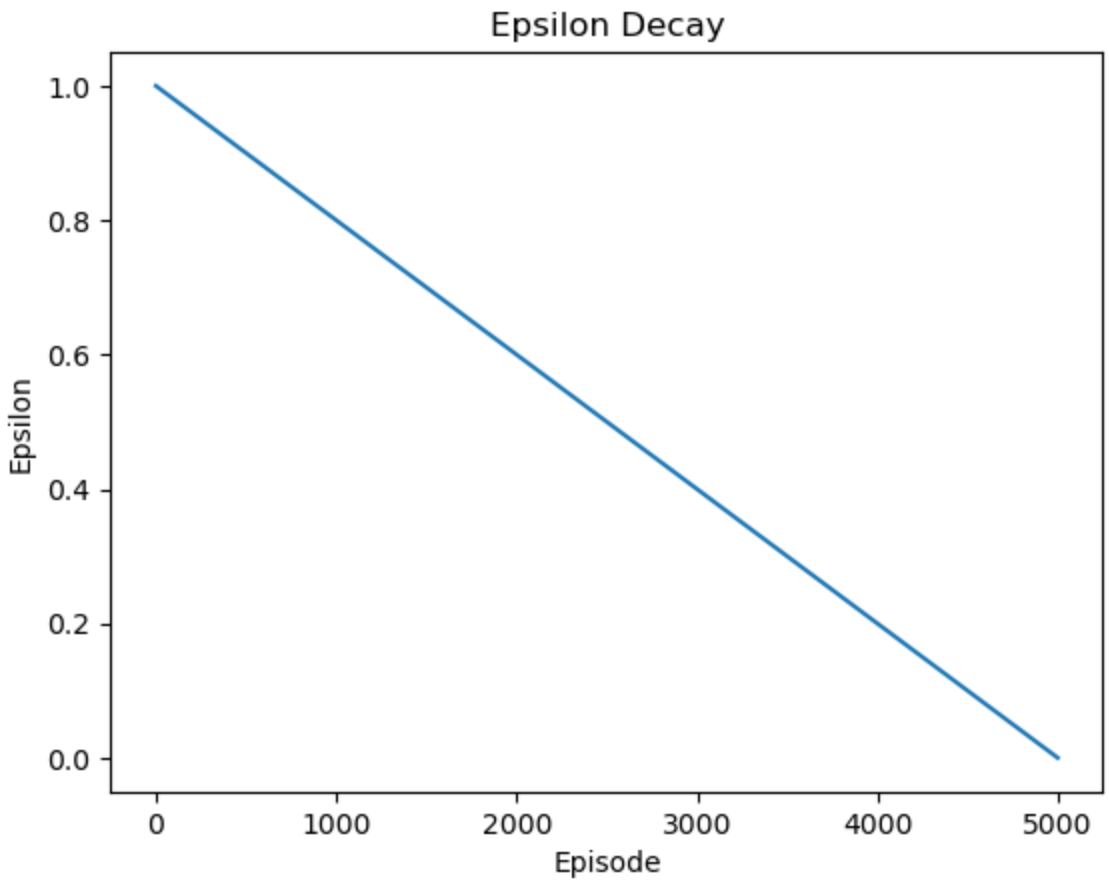
- When compared to SARSA, Q-learning can take longer to converge, particularly in situations with a high-dimensional state space.

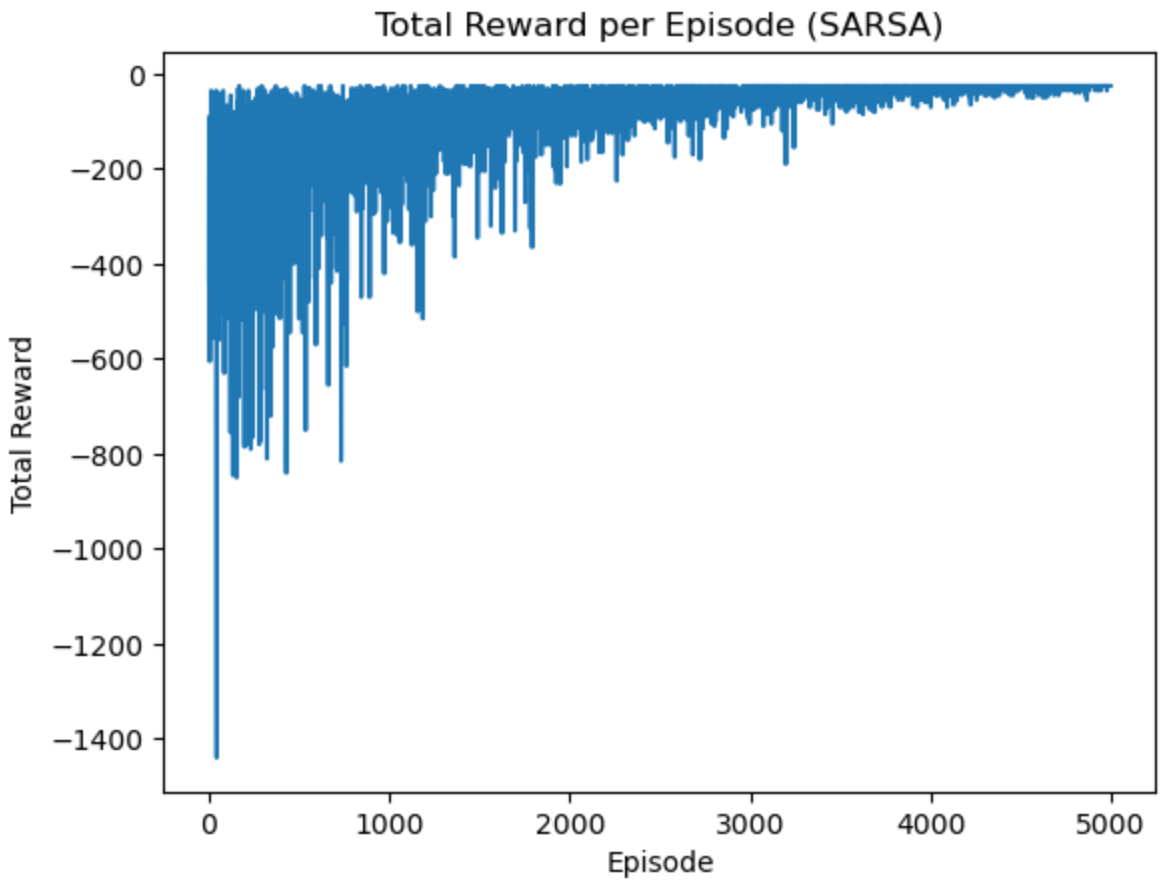
- In the case of noisy rewards or ineffective policies, it may overstate the values of state-action pairs.

2 A) The first plot displays the overall payout for each episode for various alpha, gamma, and epsilon value combinations. By experimenting with different alpha, gamma, and epsilon values, the hyperparameters were fine-tuned. The graphic demonstrates that while greater values of epsilon tend to result in lower payouts, larger values of alpha and gamma tend to result in higher prizes.

In the second plot, the epsilon value is shown to decrease over the course of the episodes. The graphic demonstrates that as the number of episodes rises, the epsilon value falls, indicating that the agent's behavior is becoming more and more predictable. This is to be expected as the agent gains experience and grows more self-assured in its abilities.

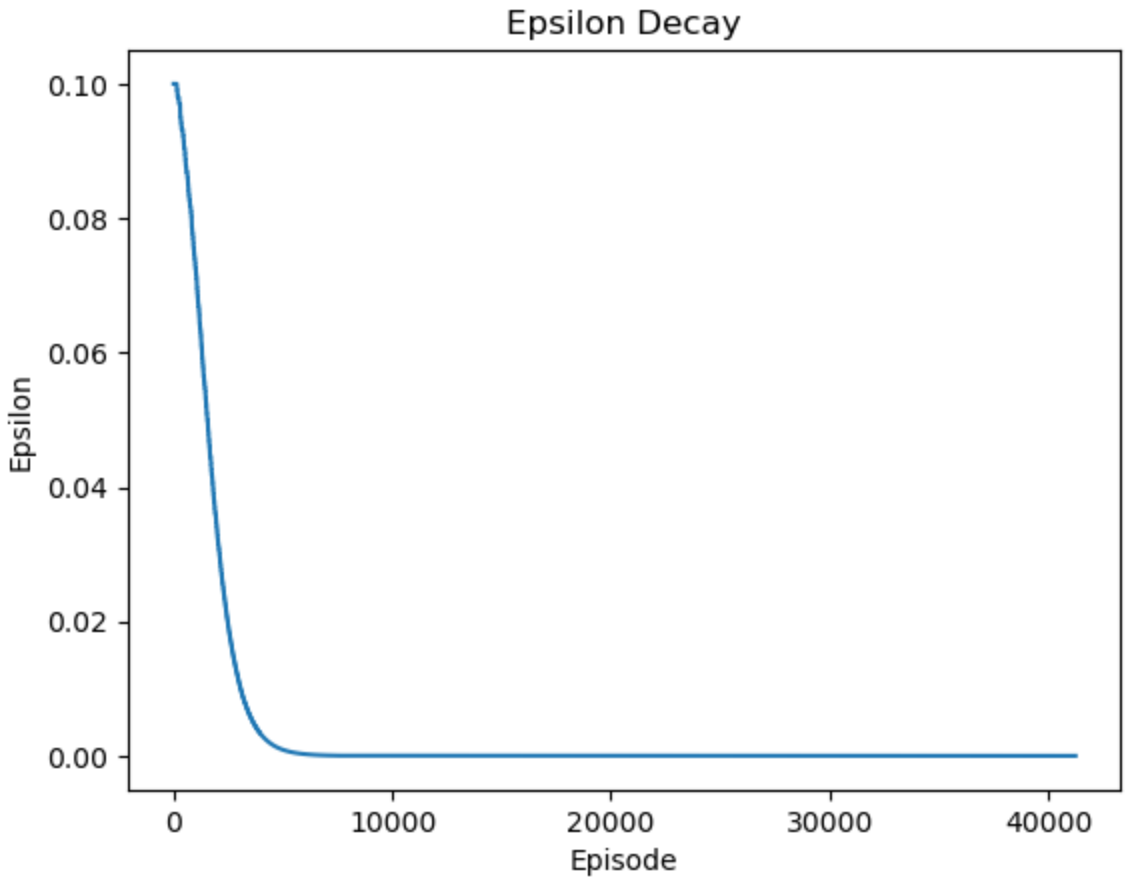
Overall, SARSA was successful in clearing the environment and earning a sizable sum of reward for each episode. To get even better results, the hyperparameters can be further tweaked.

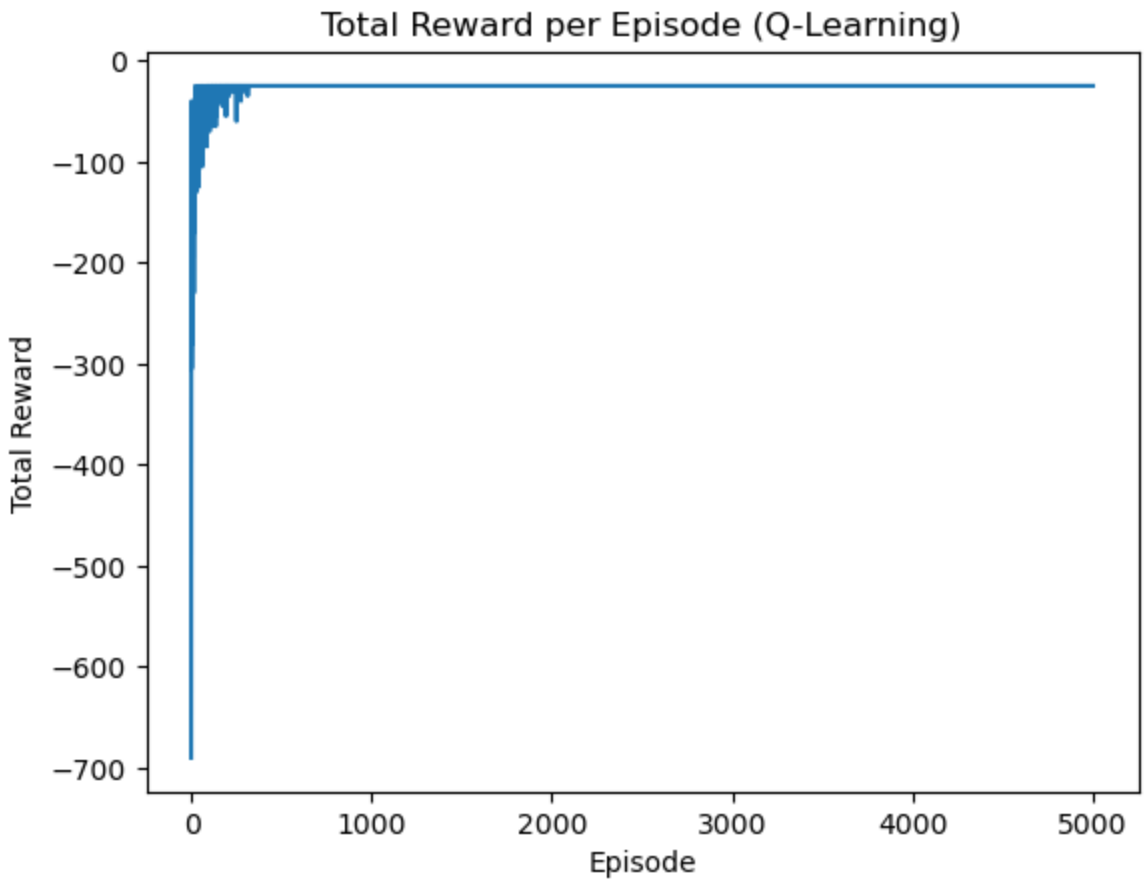

Epsilon Decay
1.0
0.8
0.6
0.4
0.2
0.0
Epsilon
0
1000
2000
3000
4000
5000
Episode



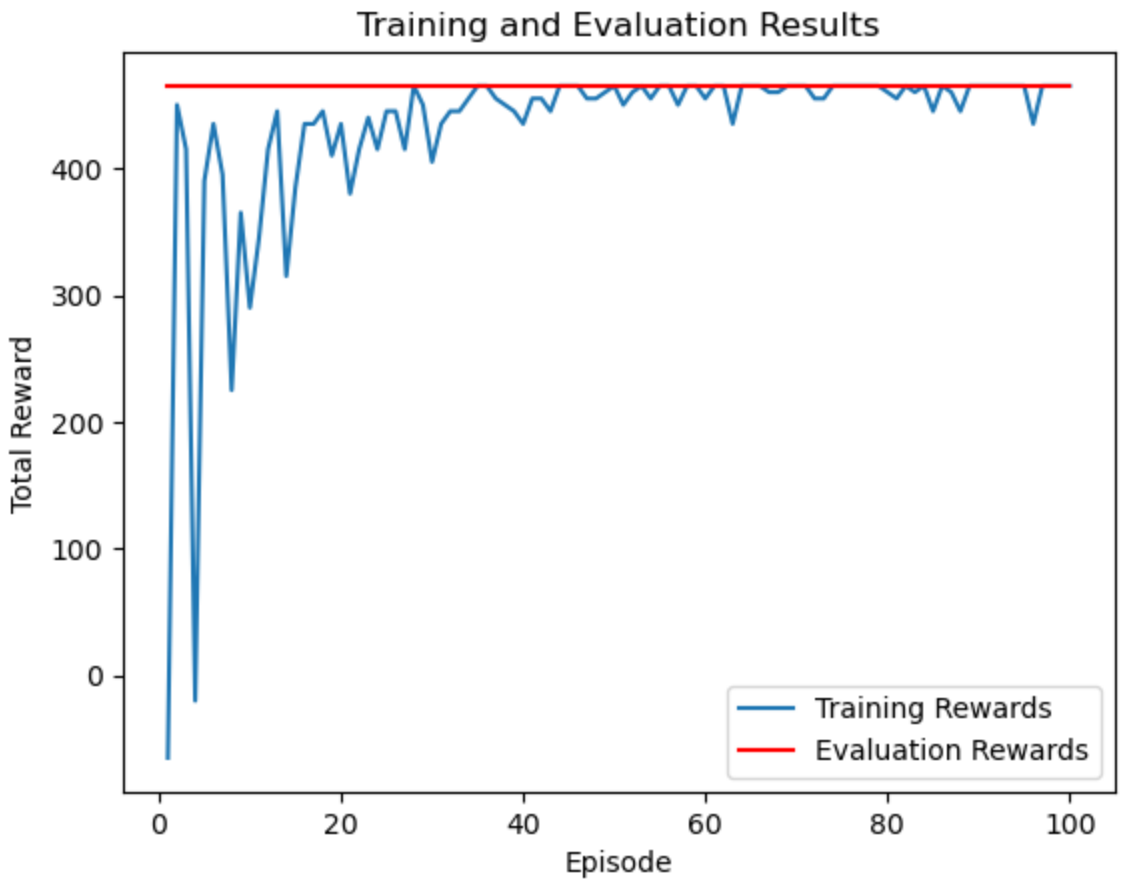
B) In order to address the Mars Exploration Rover environment described in Part 1 of this project, we used Q-learning. We trained the agent for 5000 episodes using the implementation provided in the question. In order to get the best values that minimize the negative mean reward, we varied the gamma and epsilon decay values using Optuna.

Each state-action pair's Q-value is updated by the Q-learning algorithm depending on the current reward and the discounted future reward. In this instance, we balanced exploration and exploitation using an epsilon-greedy approach. We made use of alpha's default value, which determines the size of update steps.

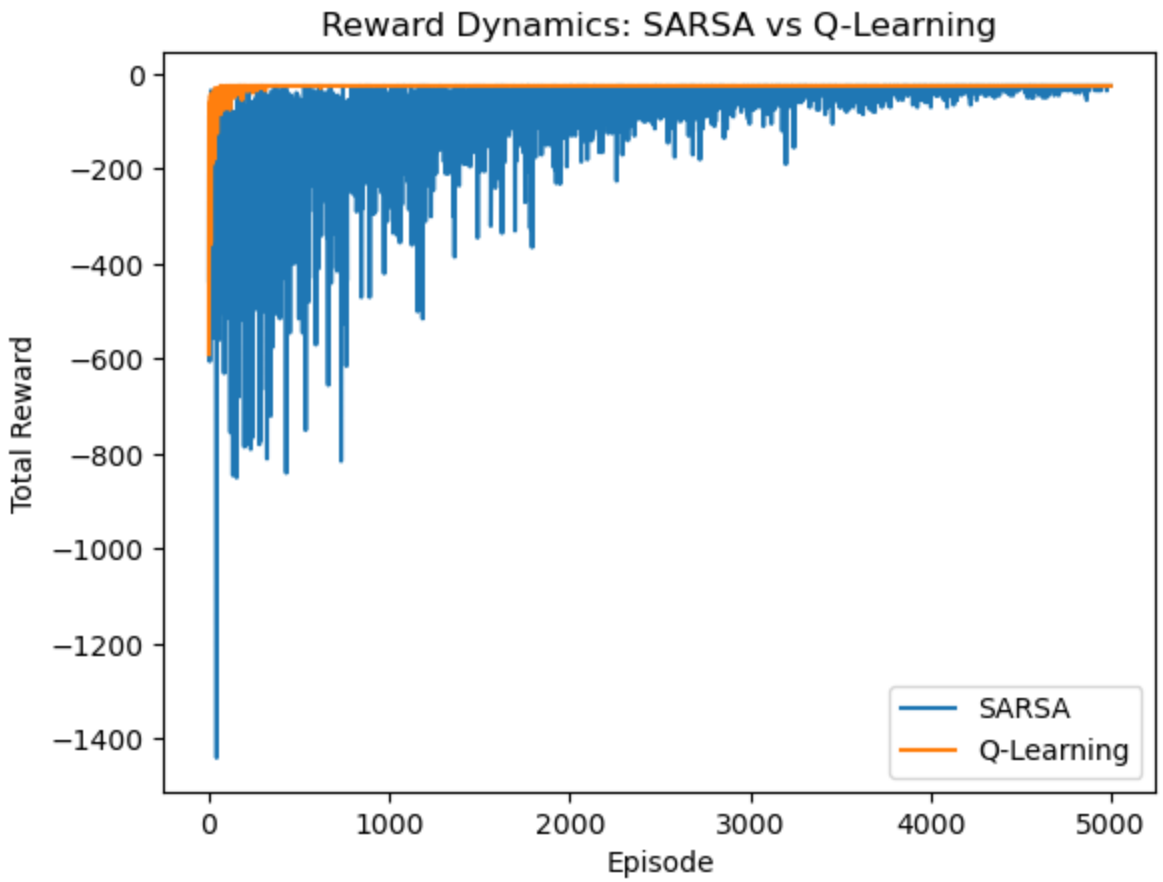

Epsilon Decay
0.10
0.08
0.06
0.04
0.02
0.00
Epsilon
0
10000
20000
30000
40000
Episode


Total Reward per Episode (Q-Learning)
Total Reward
0
−100
−200
−300
−400
−500
−600
−700
0
1000
2000
3000
4000
5000
Episode

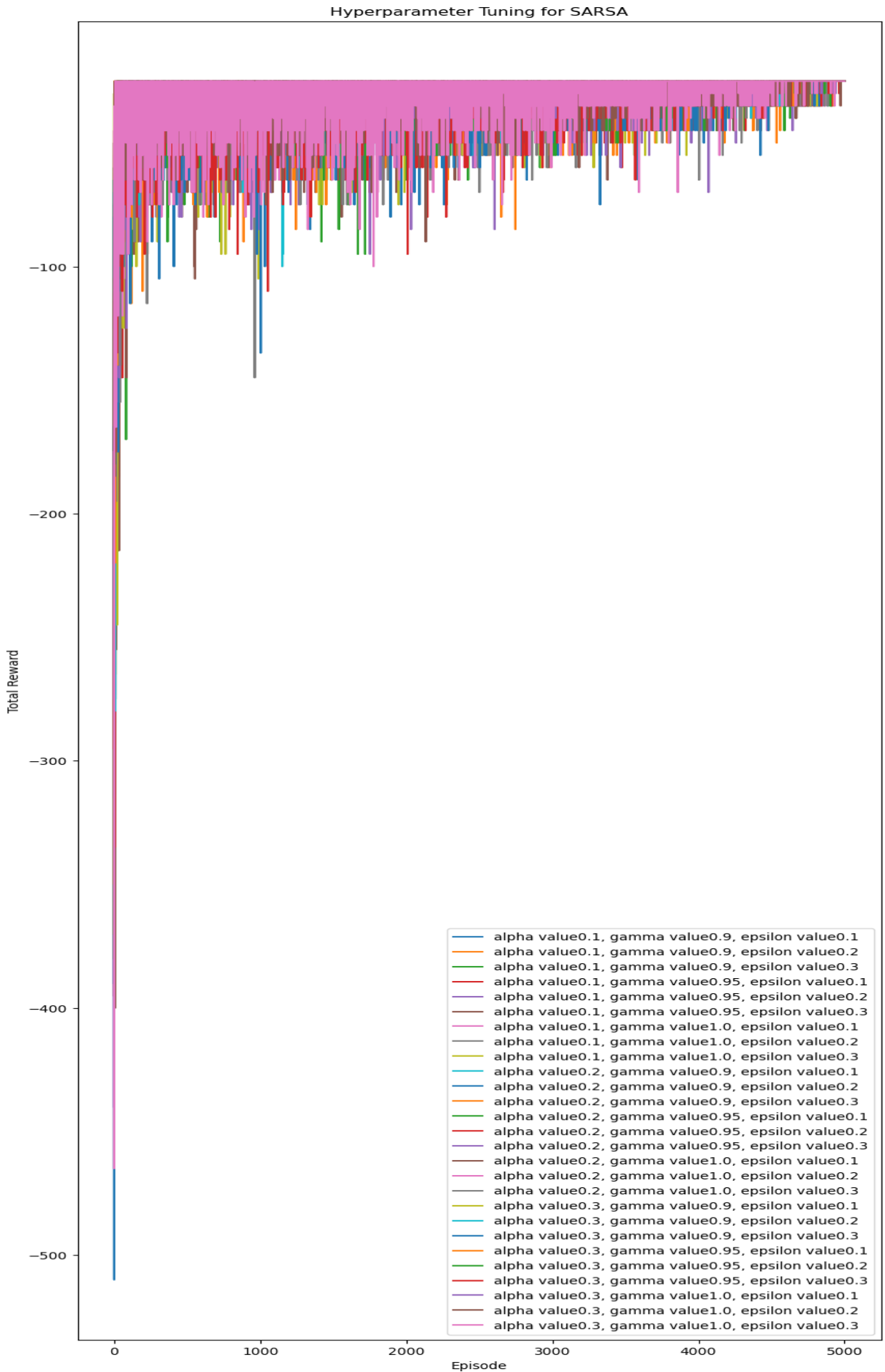
2.c)


Training and Evaluation Results
Total Reward
400
300
200
100
0
0
20
40
60
80
100
Episode
Training Rewards
Evaluation Rewards

3)

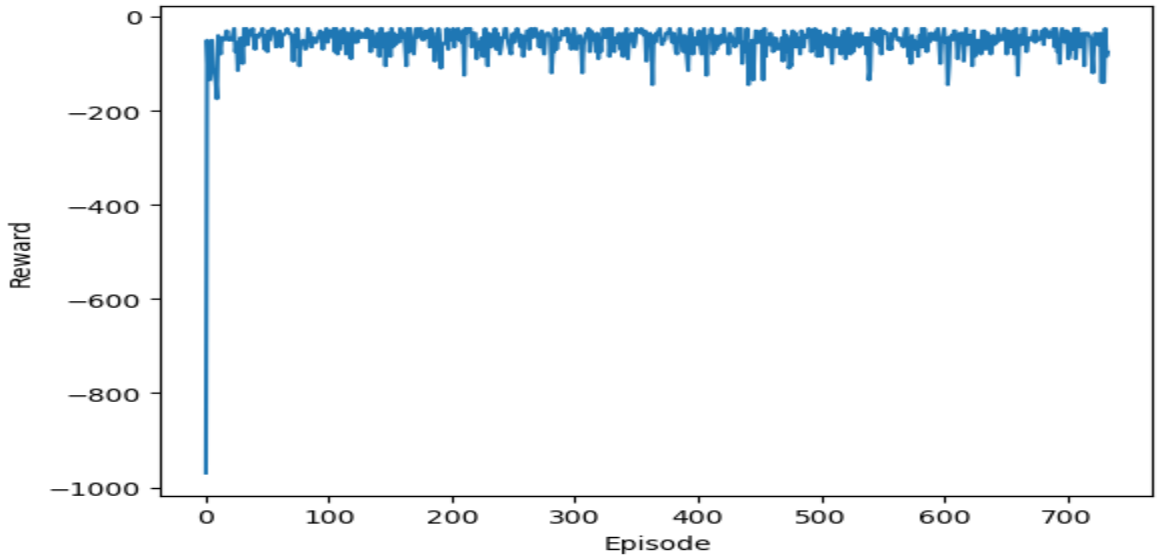

Reward Dynamics: SARSA vs Q-Learning
Total Reward
0
−200
−400
−600
−800
−1000
−1200
−1400
0
1000
2000
3000
4000
5000
Episode
SARSA
Q-Learning

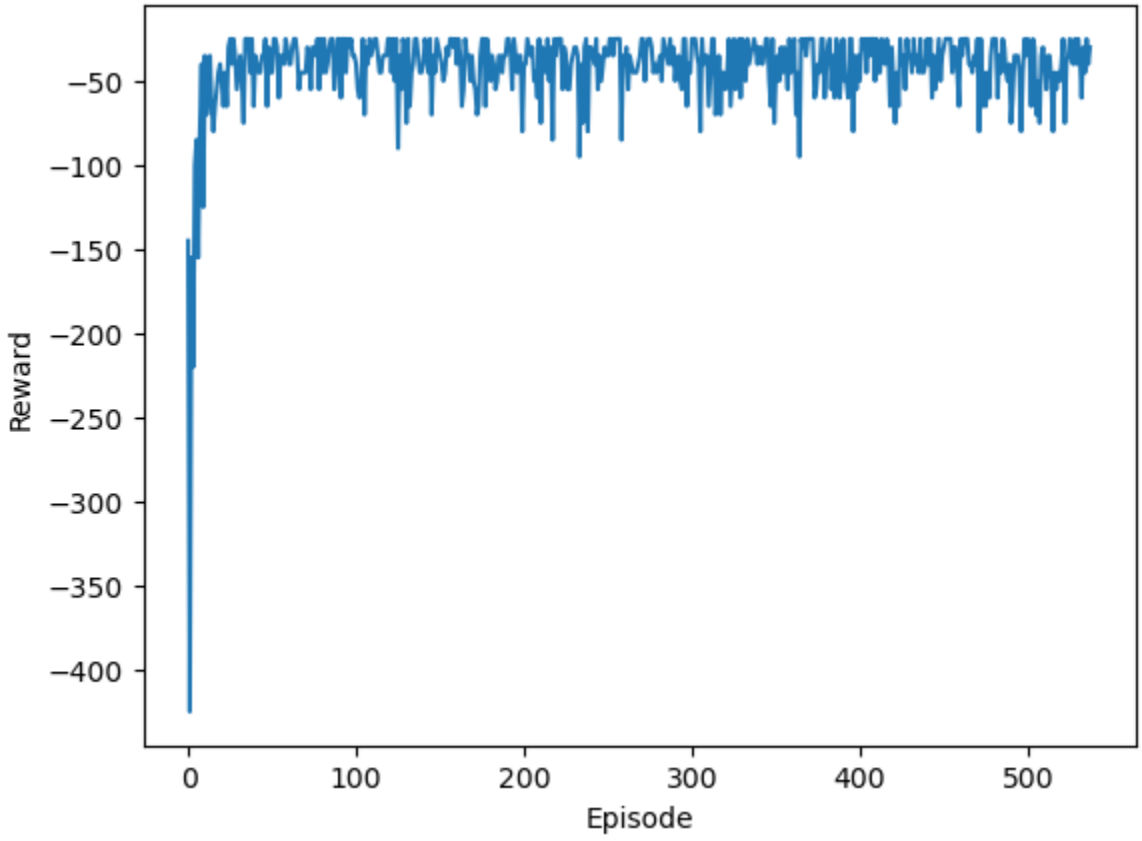
4)

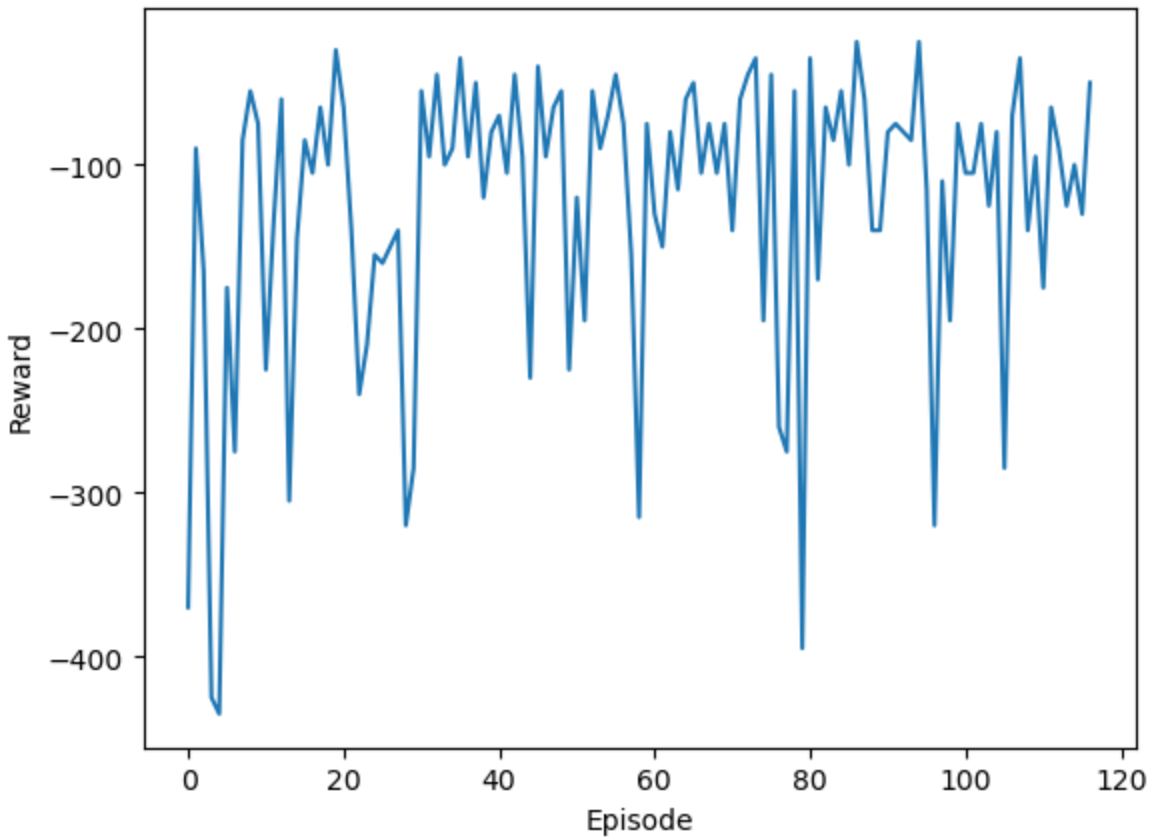

Hyperparameter Tuning for SARSA
Total Reward
−100
−200
−300
−400
−500
0
1000
2000
3000
4000
5000
Episode
alpha value0.1, gamma value0.9, epsilon value0.1
alpha value0.1, gamma value0.9, epsilon value0.2
alpha value0.1, gamma value0.9, epsilon value0.3
alpha value0.1, gamma value0.95, epsilon value0.1
alpha value0.1, gamma value0.95, epsilon value0.2
alpha value0.1, gamma value0.95, epsilon value0.3
alpha value0.1, gamma value1.0, epsilon value0.1
alpha value0.1, gamma value1.0, epsilon value0.2
alpha value0.1, gamma value1.0, epsilon value0.3
alpha value0.2, gamma value0.9, epsilon value0.1
alpha value0.2, gamma value0.9, epsilon value0.2
alpha value0.2, gamma value0.9, epsilon value0.3
alpha value0.2, gamma value0.95, epsilon value0.1
alpha value0.2, gamma value0.95, epsilon value0.2
alpha value0.2, gamma value0.95, epsilon value0.3
alpha value0.2, gamma value1.0, epsilon value0.1
alpha value0.2, gamma value1.0, epsilon value0.2
alpha value0.2, gamma value1.0, epsilon value0.3
alpha value0.3, gamma value0.9, epsilon value0.1
alpha value0.3, gamma value0.9, epsilon value0.2
alpha value0.3, gamma value0.9, epsilon value0.3
alpha value0.3, gamma value0.95, epsilon value0.1
alpha value0.3, gamma value0.95, epsilon value0.2
alpha value0.3, gamma value0.95, epsilon value0.3
alpha value0.3, gamma value1.0, epsilon value0.1
alpha value0.3, gamma value1.0, epsilon value0.2
alpha value0.3, gamma value1.0, epsilon value0.3

We are combining all three graphs for hyperparameter tuning of SARSA in one just for a clear view,
From the graph above it is clear that performance can be improved thanks to the SARSA Algorithm, from the above graph we see that an alpha of 0.3 gamma of 0.9 and epsilon of 0.3 is the most efficient setup but its likely that we could have ended up with a different results (perhaps even better?) if we had chosen different hyperparameters to train.

For the Hyperparameter tuning of Q-learning:

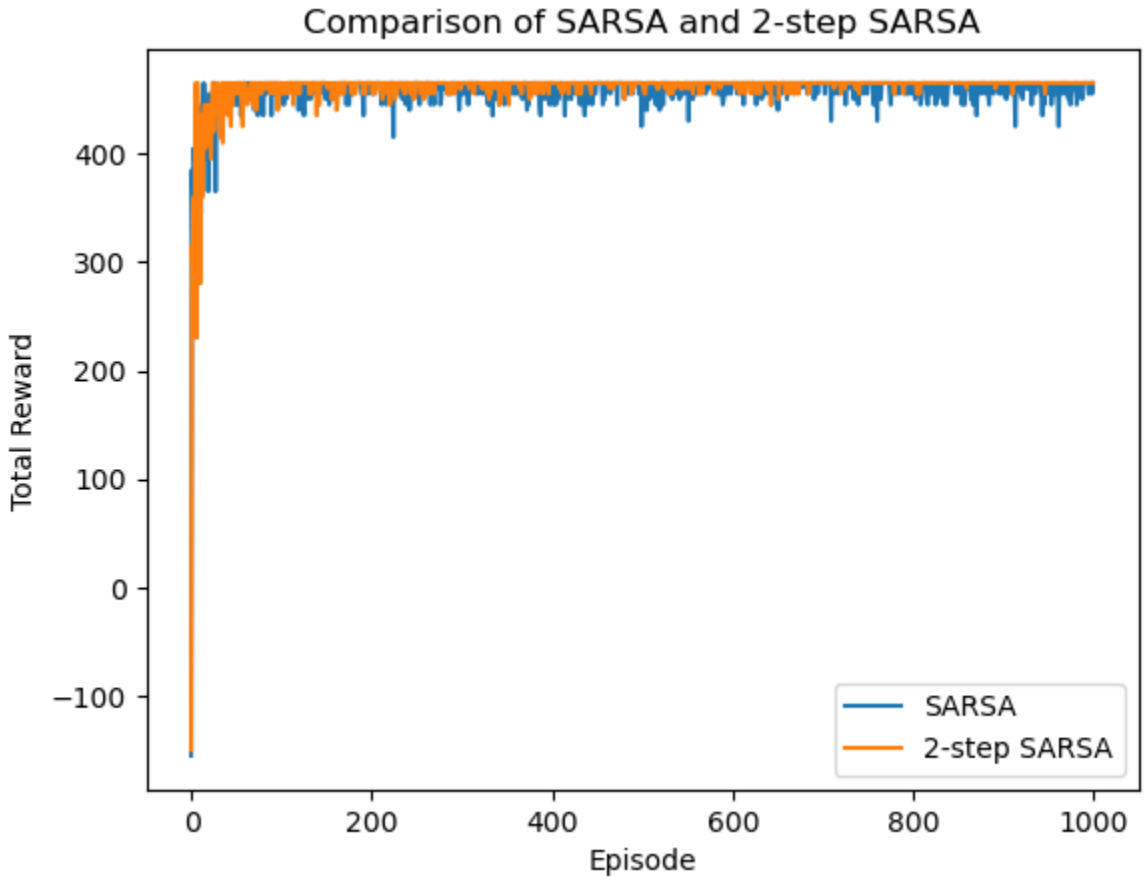

0
−200
−400
−600
−800
−1000
Reward
0
100
200
300
400
500
600
700
Episode


−50
−100
−150
−200
−250
−300
−350
−400
Reward
0
100
200
300
400
500
Episode



By observing the graphs, you can see how the reward changes over the episodes. The goal is to maximize the reward over time, indicating that the agent is learning and improving its performance in the environment. If the rewards increase steadily or reach a plateau at a high value, it indicates that the Q-learning algorithm has converged and is performing well. On the other hand, if the rewards fluctuate or remain low, it suggests that the algorithm is struggling to find an optimal policy.
In order to get the best outcome we can say that having a low value of alpha,high value for gamma, and high value of epsilon we are getting the best results.

Bonus



Team Member	Assignment Part	Contribution
Bipin	Part 1,2,3 and report	50%
Hussian	Part 1,2,3 and report	50%