Reinforcement Learning: Assignment 1

Suvidha Vidyasagar Deshpande

September 26, 2025

Part 1: Deterministic and Stochastic Environments

Deterministic Environment

States: The state is represented as a tuple (*row*, *column*, *carrying*) where *row* and *column* denote the robot's position on a 6×6 grid and *carrying* indicates whether the robot has picked up the package.

Actions: The available actions are `UP`, `DOWN`, `LEFT`, `RIGHT`, `PICKUP`, and `DROPOFF`.

Rewards: Step cost: -1; obstacle penalty: -20; pickup bonus: $+25$; delivery reward: $+100$.

Objective: Pick up a package at location $(2, 3)$ and deliver it to $(5, 5)$ with the maximum possible total reward.

Stochastic Environment

States: Same representation as in the deterministic environment: (*row*, *column*, *carrying*).

Actions: `UP`, `DOWN`, `LEFT`, `RIGHT`, `PICKUP`, and `DROPOFF`.

Randomness in Action: In the `StochasticWarehouseEnv` subclass, there is an additional 20% chance that a movement action returns -1 reward without moving.

Rewards: Step cost: -1; obstacle penalty: -20; pickup bonus: $+25$; delivery reward: $+100$.

Objective: Same as deterministic environment, but with added randomness in action outcomes.

Visualizations of the Environments

A simple textual grid visualization:

```
. . . . . .
. # # . . .
. . . P . .
. . . . # .
. . . . . .
```

`. . . . . D`

Symbols:

- "." = free space
- "#" = obstacle
- "P" = pickup point
- "D" = drop-off point
- "R" = robot ("R*" if carrying)

This visualization is printed at each time step by the `render()` method in the environment.

Definition of the Stochastic Environment

Starting from the same grid-world layout as the deterministic version, the state space and reward structure remain the same, but the transition dynamics are modified: for each action the agent takes, with an 80% probability the intended move occurs and with a 20% probability the agent "slips" to one of the other adjacent cells. This ensures that

$$\sum_{s',r} P(s', r|s, a) = 1$$

while making the outcome uncertain. The reward received is based on the actual state reached, not the intended one, so the agent must learn a robust policy under uncertainty. All other parts of the environment (action set, terminal states, boundaries) are unchanged from the deterministic case.

Difference Between Deterministic and Stochastic Environments

In a deterministic environment, given a particular state and action, the next state and reward are always the same. In a stochastic environment, the same action from the same state can lead to different next states and/or rewards with certain probabilities.

Safety in AI Environments

The following measures ensure safety in the environments:

- All actions are validated against the defined `action_space` using an assertion to reject invalid actions.
- The robot's position is clamped within grid boundaries to prevent it from leaving the defined state space.
- Obstacles are explicitly marked; entering them results in a penalty rather than undefined behavior.

- Pickup and drop-off actions only succeed at the correct coordinates and under the proper conditions (carrying or not carrying).

These constraints ensure the agent only takes allowed actions and always operates within the defined state space.

Part 2: Q Learning

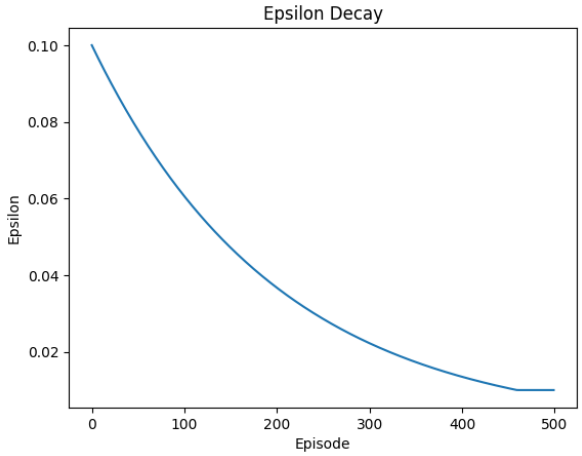

Figure 1: *
Epsilon Decay for Q-learning (Deterministic) ϵ=0.5

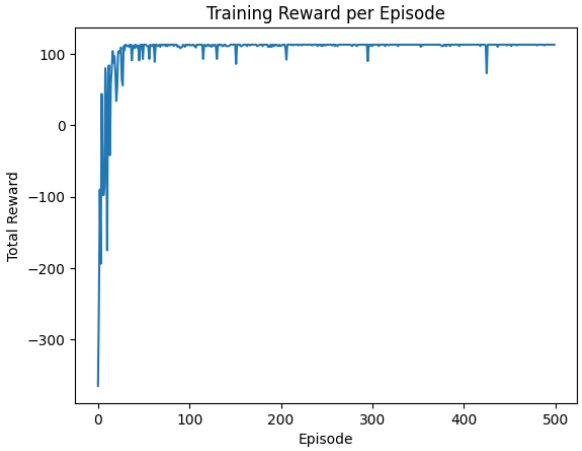

Figure 2: Total reward per episode for Q-learning (Deterministic) ϵ=0.5

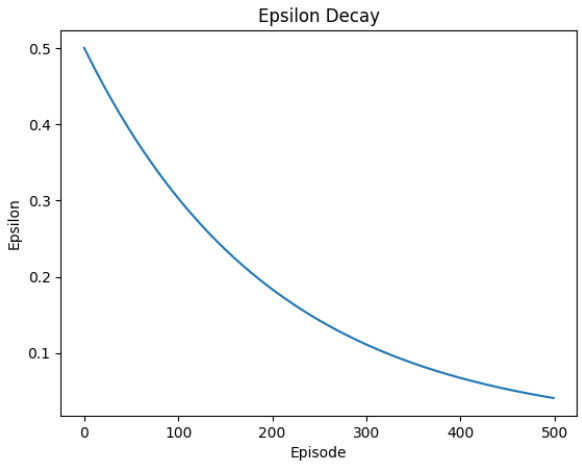

Figure 3: *
Epsilon Decay for Q-learning (Deterministic) ϵ=0.1

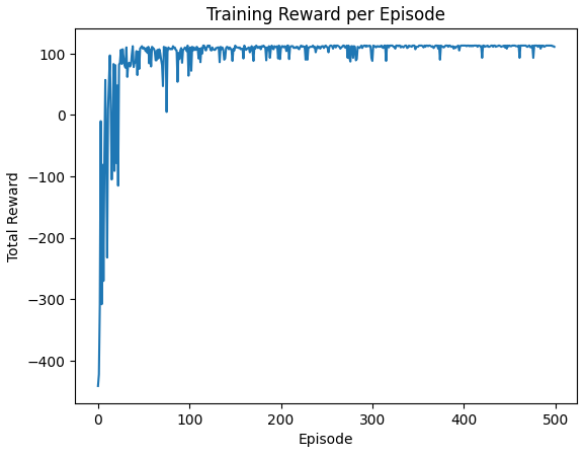

Figure 4: Total reward per episode for Q-learning (Deterministic) ϵ=0.1

Q-learning on Deterministic Environment

We evaluated different hyperparameter settings to study their effect on convergence:

- Discount factor (γ): 0.1,0.5, 0.8, 0.99
- Initial exploration rate (ϵ): 0.1, 0.5, 0.9, 0.6

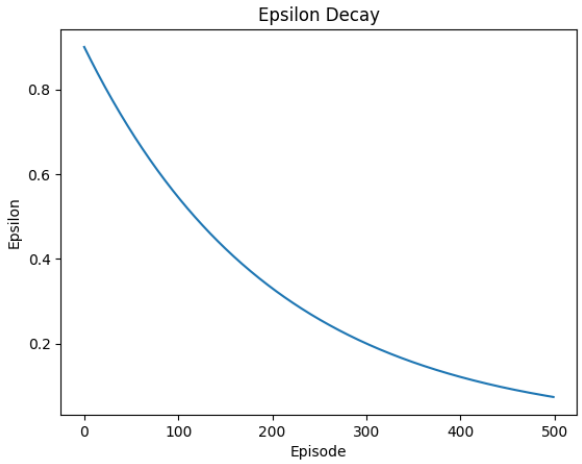

Figure 5: *
Epsilon Decay for Q-learning (Deterministic) ϵ=0.9

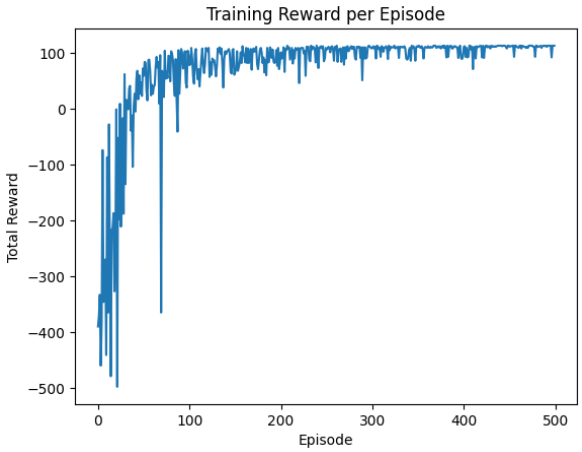

Figure 6: Total reward per episode for Q-learning (Deterministic) ϵ=0.9

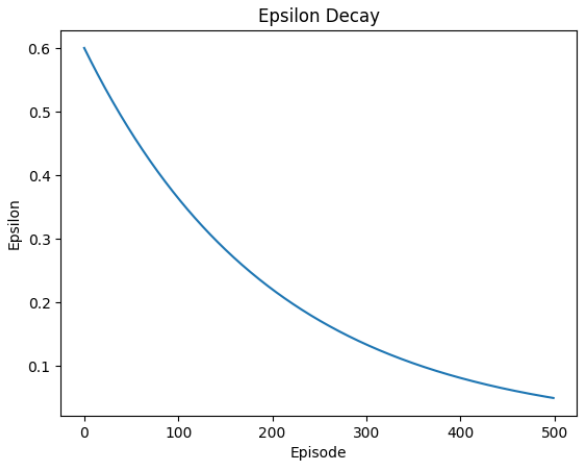

Figure 7: *
Epsilon Decay for Q-learning (Deterministic) ϵ=0.6

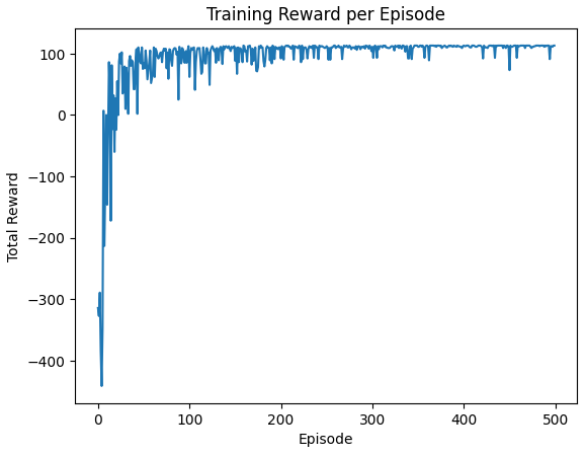

Figure 8: Total reward per episode for Q-learning (Deterministic) ϵ=0.6

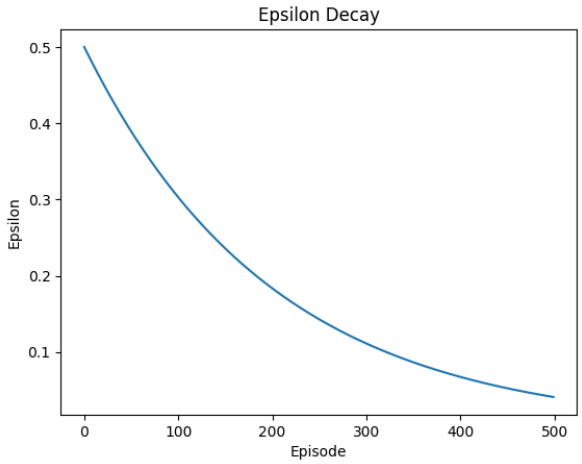

Figure 9: *
Epsilon Decay for Q-learning (Deterministic) γ=0.8

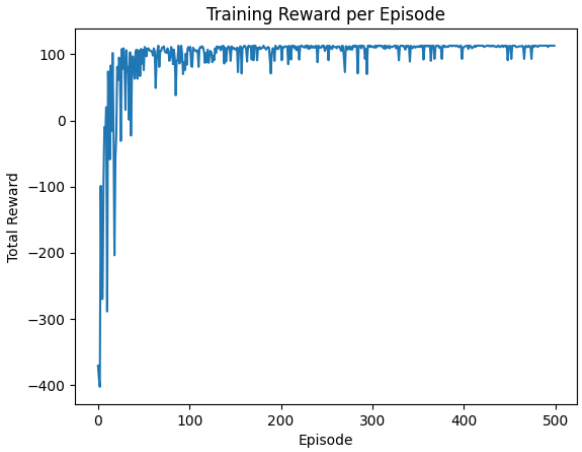

Figure 10: Total reward per episode for Q-learning (Deterministic) γ=0.8

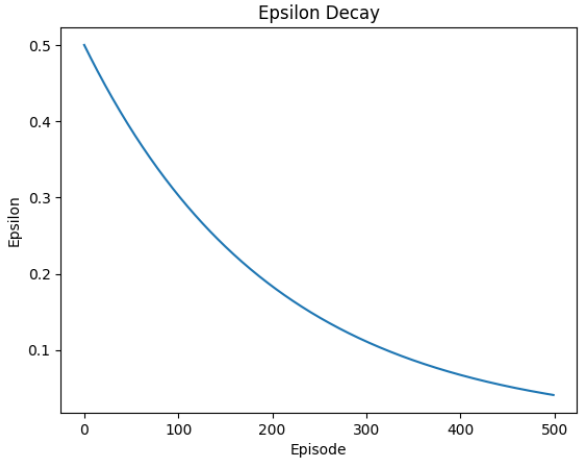

Figure 11: *
Epsilon Decay for Q-learning (Deterministic) γ=0.5

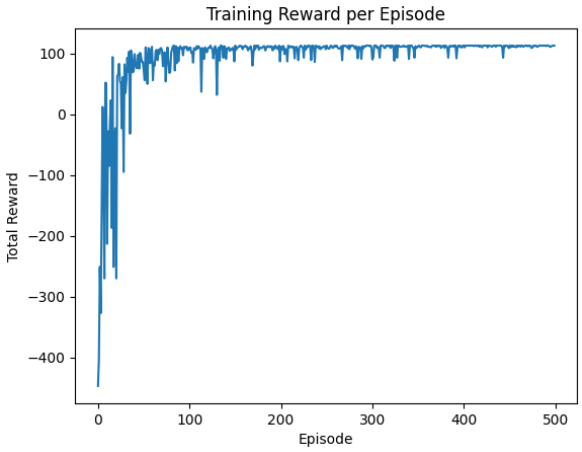

Figure 12: Total reward per episode for Q-learning (Deterministic) γ=0.5

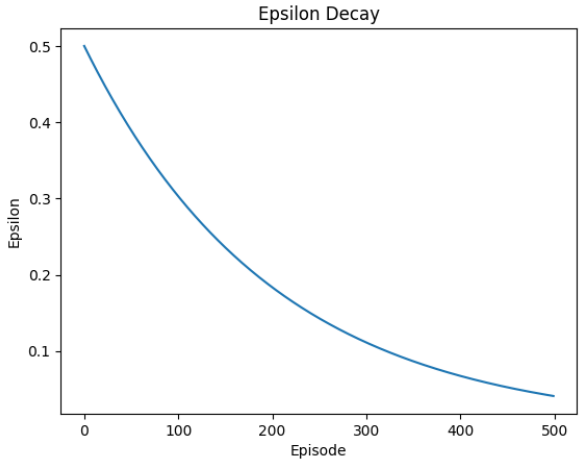

Figure 13: *
Epsilon Decay for Q-learning (Deterministic) γ=0.1

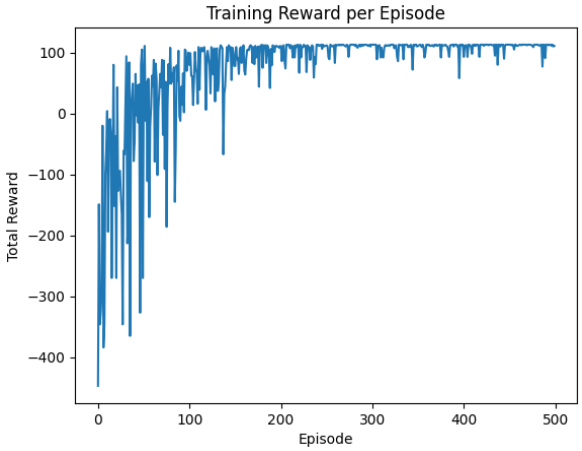

Figure 14: Total reward per episode for Q-learning (Deterministic) γ=0.1

Figures 1–14 show the results. With lower γ values, convergence is faster but final rewards plateau around +90. With $\gamma = 0.99$, the agent achieves the highest average reward ($\approx +110$) by episode 300. Similarly, extreme ϵ values (0.1 or 0.9) either under-explore or over-explore. The balanced setting $\epsilon = 0.5$ with decay rate = 0.995 provided the smoothest convergence and highest reward.

Recommendation: $\gamma = 0.99$, $\epsilon = 0.5$, decay = 0.995 are the most efficient hyperparameters for the deterministic environment.

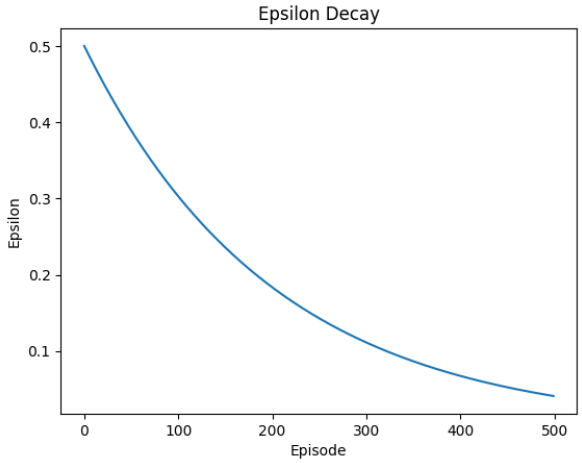

Figure 15: Epsilon Decay for Q-learning (Stochastic) ϵ=0.5

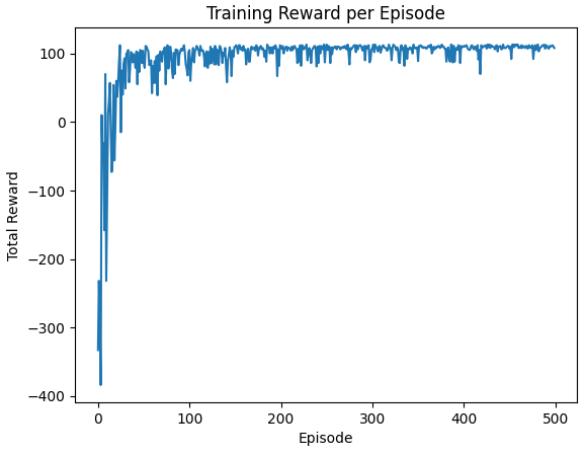

Figure 16: Total reward per episode for Q-learning (Stochastic) ϵ=0.5

Q-learning on Stochastic Environment

We repeated the tuning experiments in the stochastic environment.

- Discount factor (γ): 0.1,0.5, 0.8, 0.99
- Initial exploration rate (ϵ): 0.1, 0.5, 0.9, 0.6

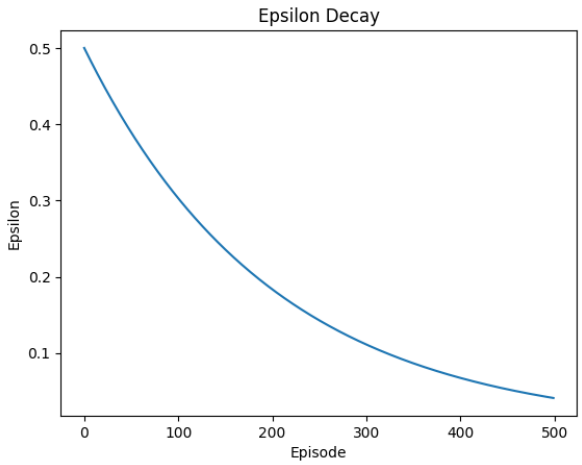

Figure 17: Epsilon Decay for Q-learning (Stochastic) ϵ=0.1

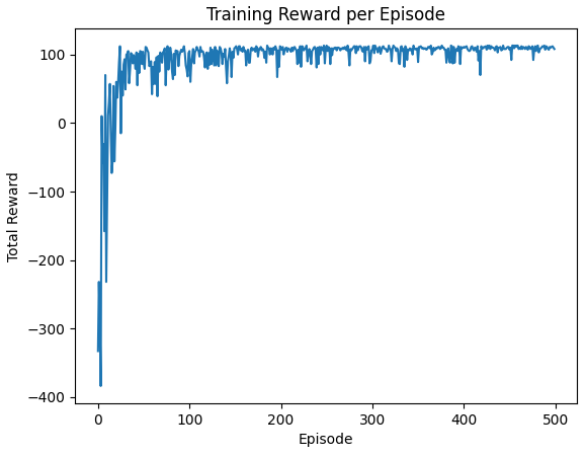

Figure 18: Total reward per episode for Q-learning (Stochastic) ϵ=0.1

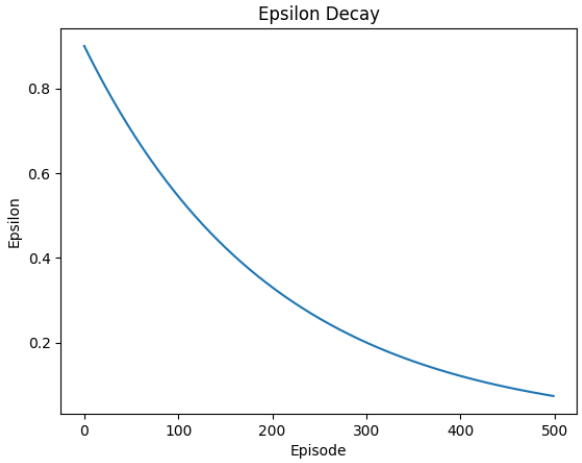

Figure 19: Epsilon Decay for Q-learning (Stochastic) ϵ=0.9

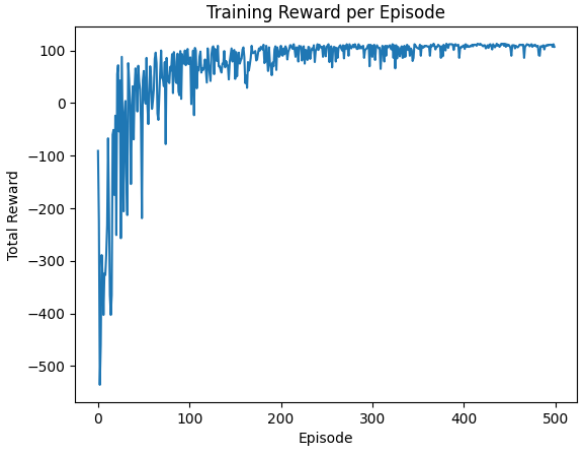

Figure 20: Total reward per episode for Q-learning (Stochastic) ϵ=0.9

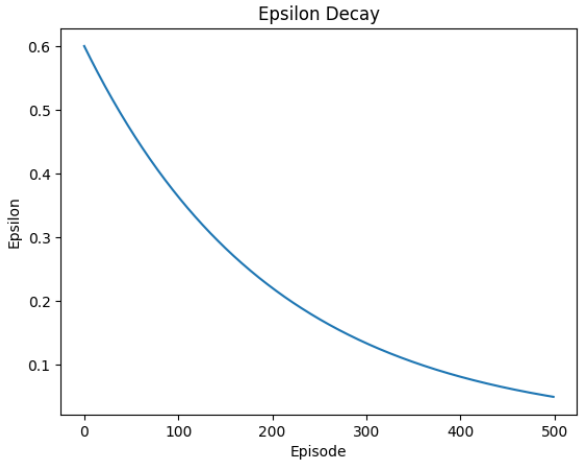

Figure 21: Epsilon Decay for Q-learning (Stochastic) ϵ=0.6

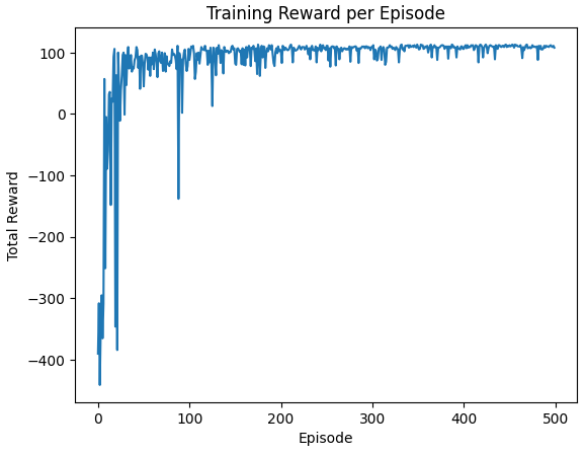

Figure 22: Total reward per episode for Q-learning (Stochastic) ϵ=0.6

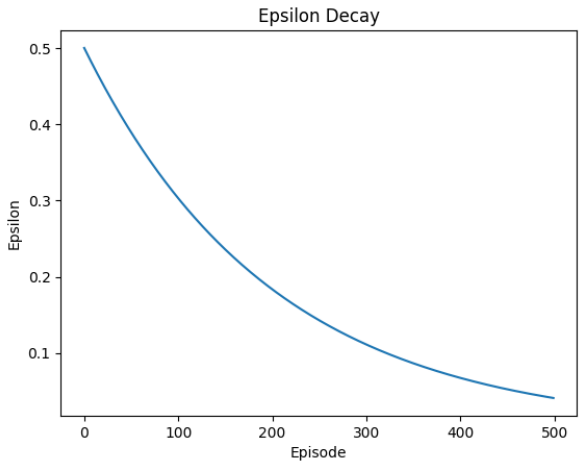

Figure 23: Epsilon Decay for Q-learning (Stochastic) γ=0.8

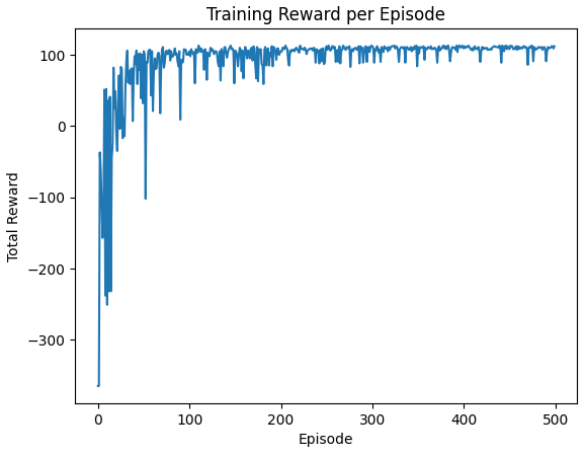

Figure 24: Total reward per episode for Q-learning (Stochastic) γ=0.8

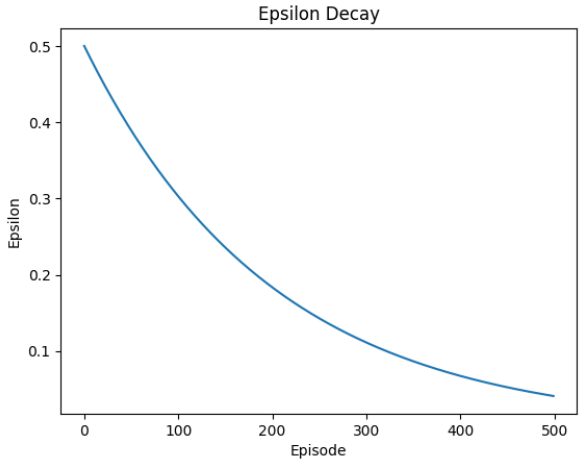

Figure 25: Epsilon Decay for Q-learning (Stochastic) γ=0.5

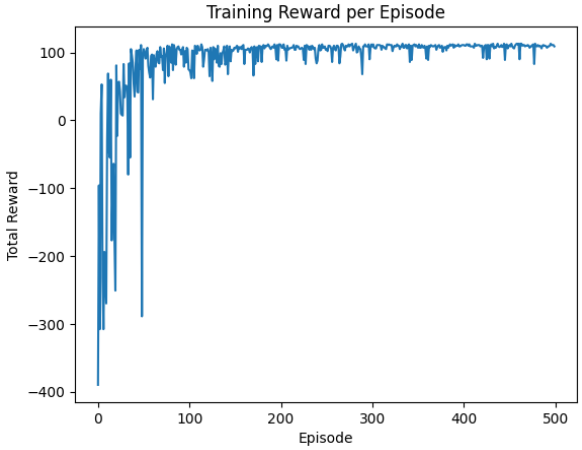

Figure 26: Total reward per episode for Q-learning (Stochastic) γ=0.5

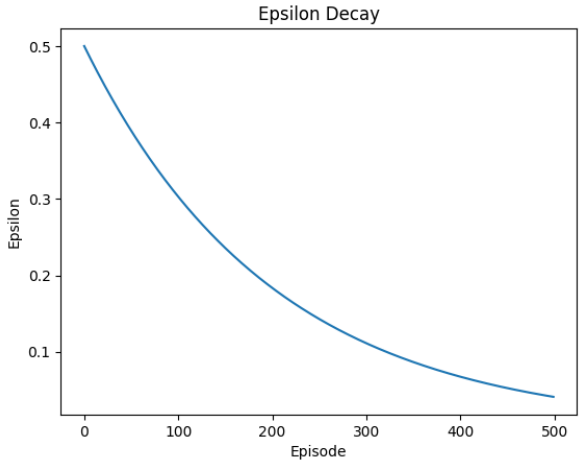

Figure 27: Epsilon Decay for Q-learning (Stochastic) γ=0.1

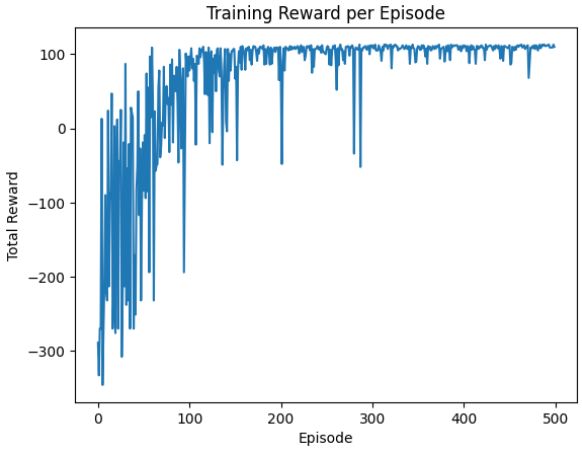

Figure 28: Total reward per episode for Q-learning (Stochastic) γ=0.1

Figures 15–28 show the results. With $\gamma = 0.95$, the agent converges more reliably, reaching average rewards around +80. $\gamma = 0.99$ leads to higher peaks but unstable performance due to randomness. Among ϵ values, 0.5 with decay = 0.995 again produced the best balance, while extreme values caused instability.

Recommendation: $\gamma = 0.95$, $\epsilon = 0.5$, decay = 0.995 gave the most consistent performance in the stochastic environment.

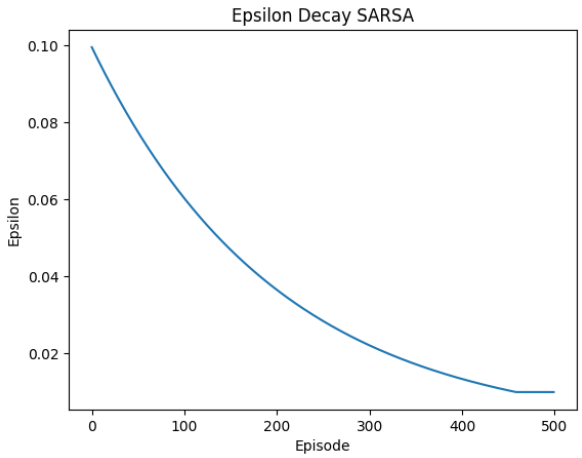

Figure 29: Epsilon Decay for SARSA(Deterministic)

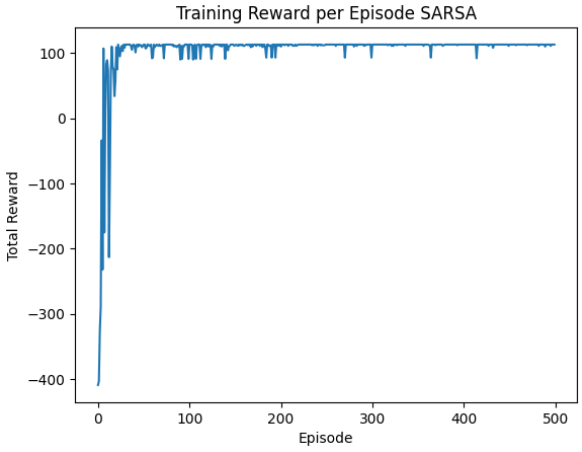

Figure 30: Total reward per episode for SARSA(Deterministic)

SARSA on Deterministic Environment

SARSA was trained with the same base model parameters:

- $\epsilon = 0.5$, $\gamma = 0.99$, decay rate = 0.995, $\epsilon_{\text{min}} = 0.01$.
- Episodes: 500.
- Steps per episode: 200.

Figures 5 and 6 show epsilon decay and total rewards. Like Q-learning, epsilon falls from 0.5 to 0.01, but the reward curve grows more conservatively: starting near −60 and stabilizing at about +100 by episode 400. SARSA's on-policy updates account for exploratory actions, making it safer but slower to converge than Q-learning.

SARSA on Stochastic Environment

Hyperparameters remain the same as above:

- $\epsilon = 0.5$, $\gamma = 0.99$, decay rate = 0.995, $\epsilon_{\text{min}} = 0.01$.
- Episodes: 500.
- Steps per episode: 200.

Figures 31 and 32 illustrate SARSA's performance. Rewards begin near −80 and gradually improve, stabilizing around +70 to +90 by the end of training. Compared to Q-learning , SARSA's curve is more stable but slower to reach high values. This highlights its robustness in noisy environments.

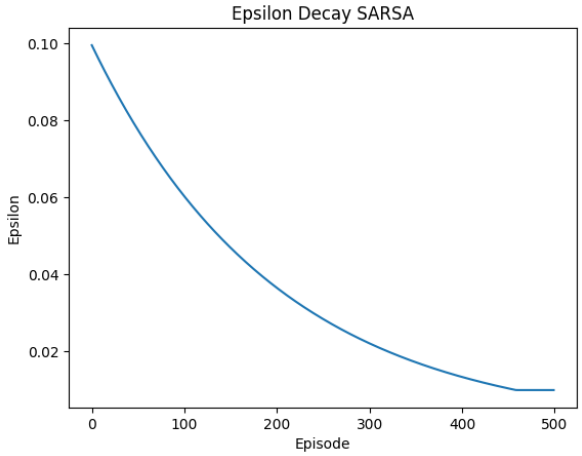

Figure 31: Epsilon Decay for SARSA(Stochastic)

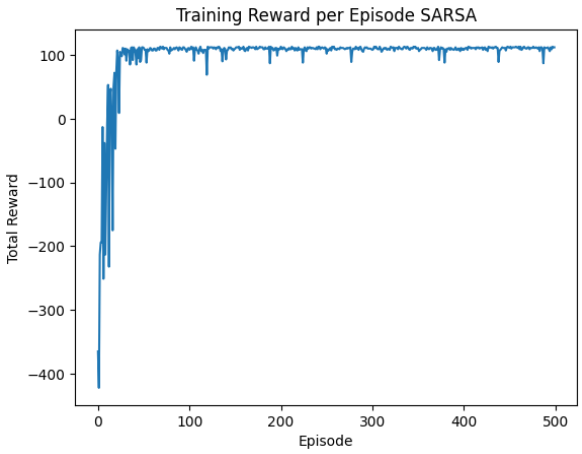

Figure 32: Total reward per episode for SARSA(Stochastic)

Figure 33: Q-learning Greedy Policy for 10 episodes (Stochastic)

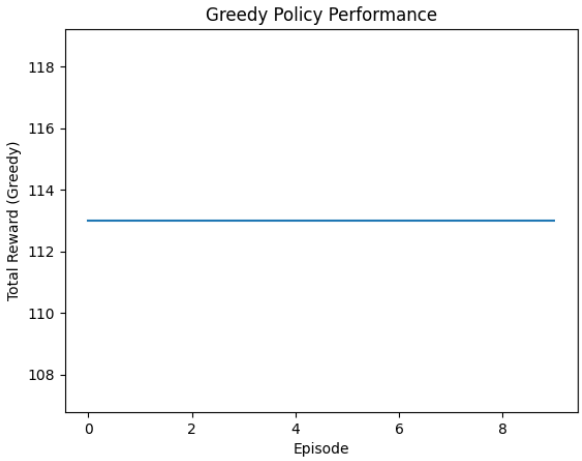

Figure 34: Q-learning Greedy Policy for 10 episodes(Deterministic)

Greedy Policy Evaluation

Figures 33 and 34 show greedy evaluation for 10 episodes using Q-learning policies. In the stochastic environment (Figure 33), rewards per episode range between 108 and 113, while in the deterministic environment (Figure 34), the agent consistently achieves about 113. This confirms the reliability of the learned policies when executed without exploration.

Rendered Greedy Episode

One greedy episode was rendered step by step. video (attached separately) show the robot navigating toward the pickup location, successfully collecting the package, avoiding obstacles, and delivering it to $(5, 5)$. The final total reward for this episode was approximately $+113$, validating the correctness of the policy.

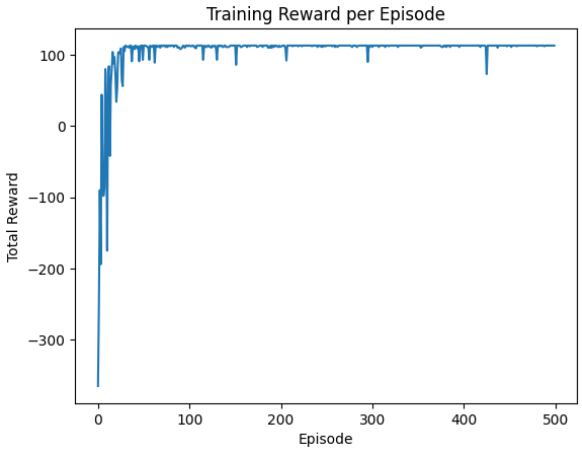

Figure 35: Qlearning on Deterministic Env

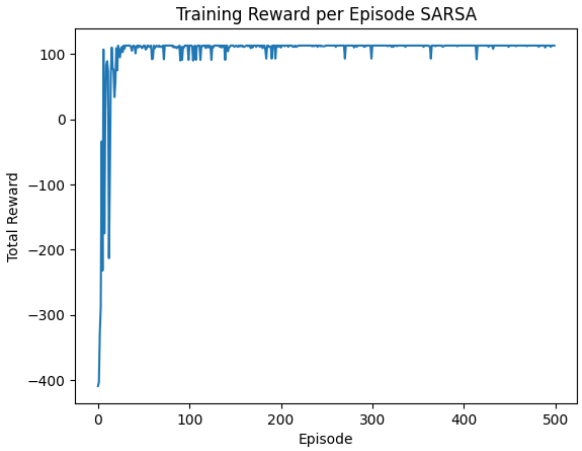

Figure 36: SARSA on Deterministic Env

Comparison in Deterministic Environment

Figures 35 and 36 compare Q-learning and SARSA in the deterministic environment. Q-learning (Figure 35) converges faster, reaching $+100$ by episode 250, while SARSA (Figure 36) converges around episode 400. Both solve the task, but Q-learning benefits from its off-policy nature by accelerating convergence.

Comparison in Stochastic Environment

Figures 37 and 38 compare Q-learning and SARSA in the stochastic environment. Q-learning (Figure 13) fluctuates between $+40$ and $+100$ due to overestimation under randomness, whereas SARSA (Figure 38) stabilizes more reliably between $+70$ and $+90$. This shows SARSA's advantage in handling uncertainty.

Tabular Methods Summary

Q-learning:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

- Off-policy, fast convergence in deterministic environments.

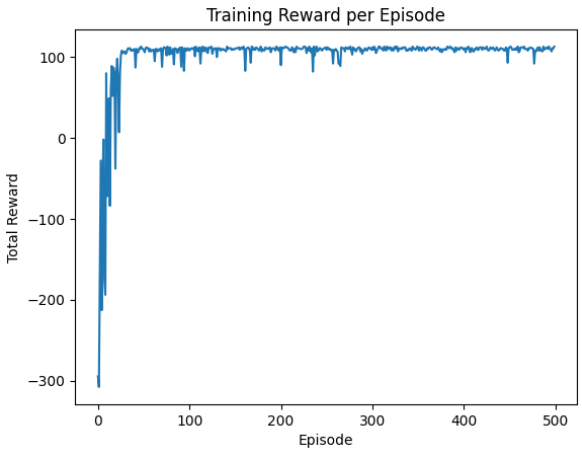

Figure 37: Qlearning on Stochastic Env

Figure 38: SARSA on Stochastic Env

- May overestimate in stochastic environments.

SARSA:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma Q(s',a') - Q(s,a)]$$

- On-policy, more stable in stochastic environments.
- Slower convergence than Q-learning.

Double Q-learning: Double Q-learning maintains two separate Q-tables, Q_1 and Q_2, to reduce overestimation. The update rules are:

$$\text{If updating } Q_1: \quad Q_1(s,a) \leftarrow Q_1(s,a) + \alpha\Big[r + \gamma Q_2(s', \arg\max_{a'} Q_1(s',a')) - Q_1(s,a)\Big]$$

$$\text{If updating } Q_2: \quad Q_2(s,a) \leftarrow Q_2(s,a) + \alpha\Big[r + \gamma Q_1(s', \arg\max_{a'} Q_2(s',a')) - Q_2(s,a)\Big]$$

- Off-policy: combines two Q-tables to reduce overestimation bias.
- More stable in stochastic environments than standard Q-learning.

Monte Carlo (Every-Visit MC): Monte Carlo control updates Q-values using returns observed at the end of episodes. Let G_t be the return following state-action pair (s_t, a_t):

$$Q(s_t,a_t) \leftarrow Q(s_t,a_t) + \frac{G_t - Q(s_t,a_t)}{N(s_t,a_t)}$$

where $N(s_t, a_t)$ is the number of times (s_t, a_t) has been visited so far.

- On-policy: uses actual episode experience to update action-values.
- Converges slowly but provides unbiased estimates.

Reward Function Design

The reward function provides intermediate guidance:

- Step cost -1.
- Obstacle penalty -20.
- Pickup bonus $+25$.
- Delivery reward $+100$.

This shaping reduces sparsity and accelerates learning. Without intermediate rewards, convergence would be much slower.

Part 3: Q-learning for Stock Trading Problem

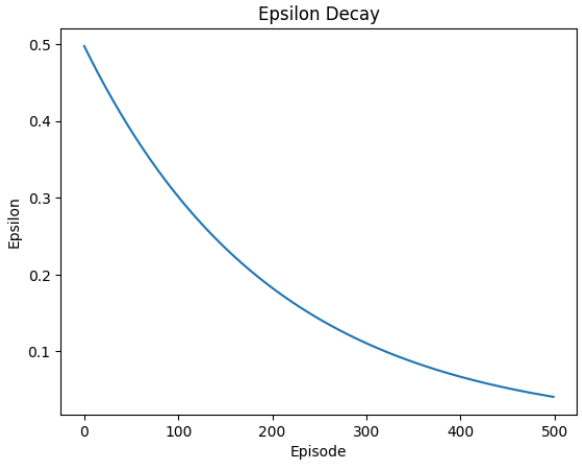

Figure 39: Epsilon decay during Q-learning training (Stock Trading)

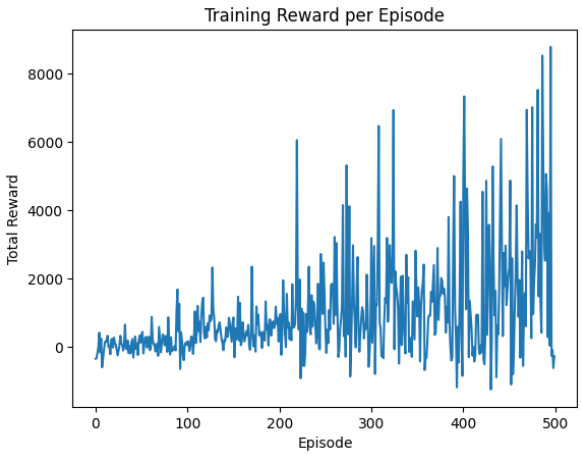

Figure 40: Total reward per episode during Q-learning training (Stock Trading)

Training Results

The Q-learning algorithm was applied to the stock trading environment with the same principles as in Part II. Figure 39 shows the epsilon decay, where the exploration rate decreases steadily from its initial value to the minimum threshold, shifting the agent from exploration to exploitation. Figure 40 illustrates the total reward per episode.Although the rewards remain highly volatile due to the stochastic and non-stationary nature of the trading environment, the overall trend shows increasing reward peaks over episodes. This indicates that the agent is learning strategies that can yield higher returns, even if performance fluctuates episode to episode

Evaluation Results

After training, the agent was evaluated with exploration turned off ($\epsilon = 0$, purely greedy policy). This means actions are selected only from the learned Q-values without random exploration.

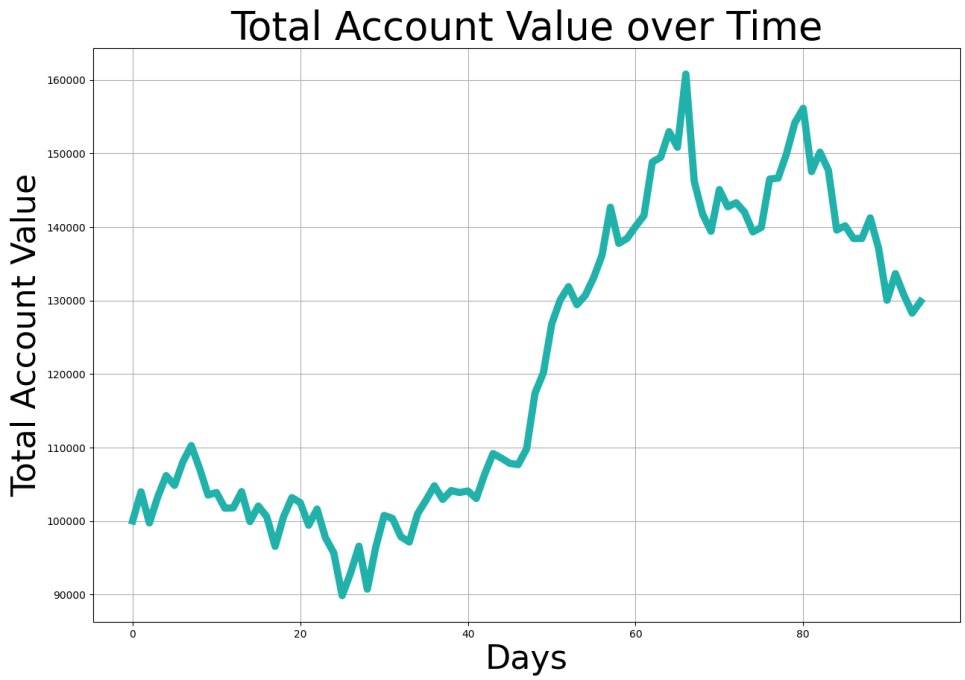

Figure 41: Agent's account value over time under greedy policy evaluation

Figure 41 shows the evolution of the agent's account value across time steps. The account value increases as the agent takes profitable trading actions and avoids losses, demonstrating that the learned policy generalizes well when evaluated greedily.

The training and evaluation results highlight several key points:

- Q-learning successfully adapts to the stock trading environment, despite noisy and non-stationary dynamics.
- The ϵ-greedy strategy with decay allows sufficient exploration initially, while gradually focusing on exploitation for stronger performance.
- Training rewards remain volatile, reflecting the uncertainty of financial markets, but the increasing peaks demonstrate learning progress.
- The evaluation plot confirms that the learned policy produces steady account value growth, showing the agent's ability to generalize after training.