**MSDS 684 – Reinforcement Learning**

**Lab 4 – Temporal Difference Learning (SARSA vs Q-Learning)**

## Section 1: Project Overview

This lab investigates two core Temporal Difference (TD) control algorithms SARSA and Q-learning in the CliffWalking-v1 environment. The goal is to understand how on-policy and off-policy learning lead to very different behaviors, even when both methods use the same state representation, reward structure, and exploration strategy. The assignment focuses on TD(0) updates, online learning, and the effect of ε-greedy exploration during training.

Temporal Difference learning sits between Monte Carlo and Dynamic Programming. Unlike Monte Carlo, TD does not wait for the full episode to finish; it updates value estimates step-by-step using bootstrapping. Unlike DP, TD does not need the full transition model. According to Sutton & Barto (Chapter 6), TD methods are effective in online settings because they update quickly and can learn directly from interaction with the environment.

The CliffWalking environment is a 4×12 grid with 48 discrete states and 4 actions (up, right, down, left). The start state is at the bottom-left corner, and the goal is at the bottom-right corner. The "cliff" cells in between cause the agent to fall, receive −100 reward, and restart the episode. All other actions give −1 reward. The episode ends when the agent reaches the goal or steps into the cliff. Because of this layout, the environment naturally highlights safe vs. risky behavior.

Based on theory, I expected the following:

- SARSA would learn a safer but longer path, because it updates based on the action it actually takes under ε-greedy exploration.
- Q-learning would learn the optimal shortest path along the cliff edge, but would be more unstable during training because it uses the max over next-state actions, ignoring exploration.
- Q-learning would likely have larger swings in return early on because exploratory moves near the cliff cause repeated failures.

This lab uses both algorithms over 30 random seeds to reduce randomness. Each algorithm was trained for 500 episodes, using ε-greedy with decay (1.0 → 0.05). The results include learning curves with 95% confidence intervals, value function heatmaps, policy arrow maps, and sample trajectories. I also ran extra experiments on learning rate α, exploration schedules, and a greedy-only Q-learning run to show what happens when exploration is removed.

Overall, this lab demonstrates how algorithm design choices mainly whether updates use the behavior policy or the greedy target policy, directly affect safety, convergence speed, and stability in TD learning.

**Section 2: Deliverables**

**GitHub Repository:**

**Lab4 link -**
https://github.com/nikhsnona/MSDS_684_ReinforcementLearning/tree/main/Lab4

**clone link - https://github.com/nikhsnona/MSDS_684_ReinforcementLearning.git**

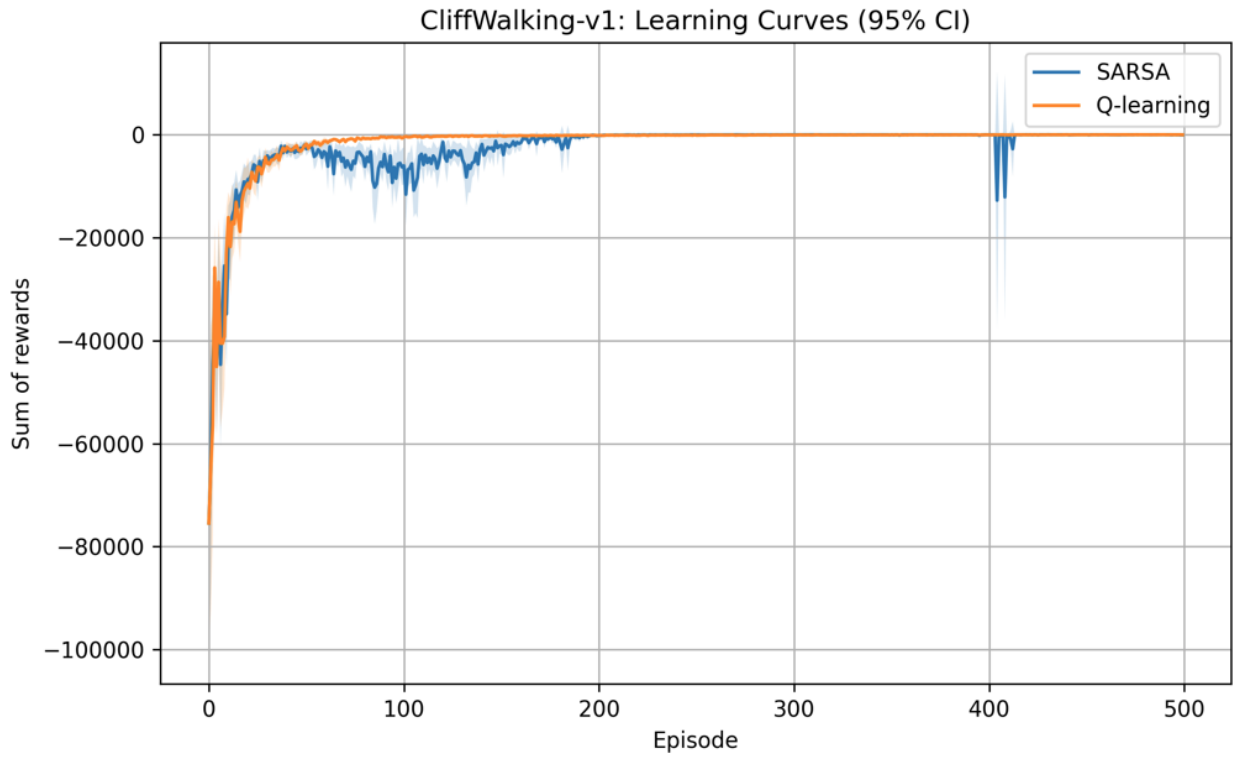
**Implementation Summary**

I implemented SARSA and Q-learning using tabular TD(0) control in the CliffWalking-v1 environment. Both algorithms use ε-greedy exploration with decay from 1.0 to 0.05. The Q-table has shape (48 states × 4 actions). TD updates occur immediately after every environment step. For SARSA, the TD target uses the value of the next action actually selected; for Q-learning, the target uses the maximum action value for the next state. Each algorithm ran for 500 episodes across 30 seeds. I generated learning curves, value heatmaps, policy arrow plots, trajectories, and additional experiments on α values, exploration schedules, and a greedy-only Q-learning case. All figures were saved in the figures_lab4 folder.

**Key Results & Analysis**

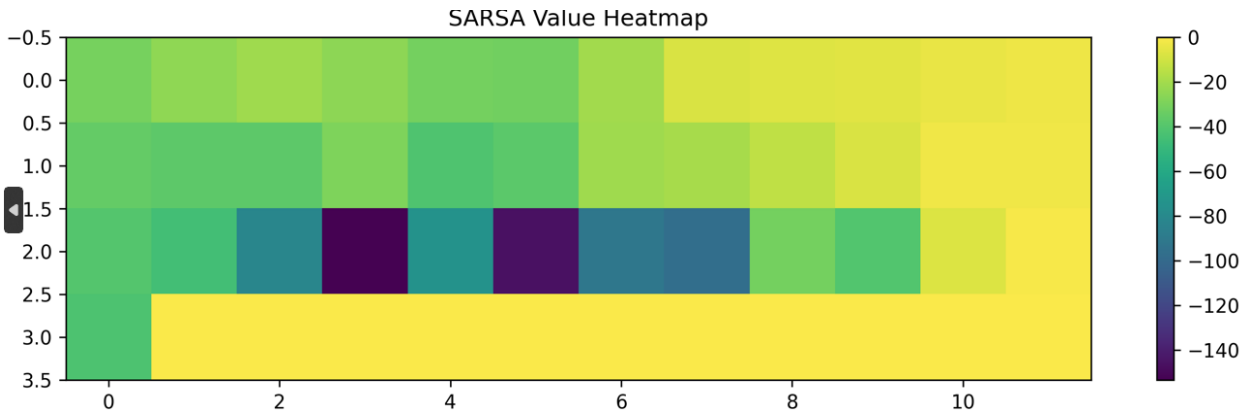
**1. Learning Performance**

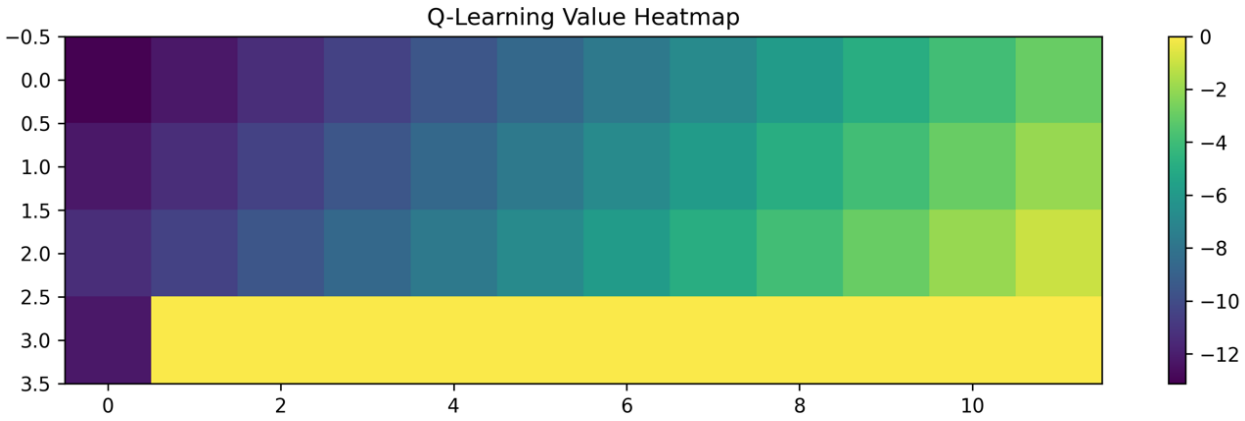
The learning curves show that both algorithms improved steadily from very large negative returns early in training to values close to zero by the end. The final averaged returns were −43.92 for SARSA and −41.75 for Q-learning, indicating that Q-learning performed slightly better numerically, though both methods showed similar long-run behavior. The “EpTo90%” value was reported as *nan* because neither algorithm reached 90% of its best mean reward consistently across all 30 seeds.




**2. Value Heatmaps**

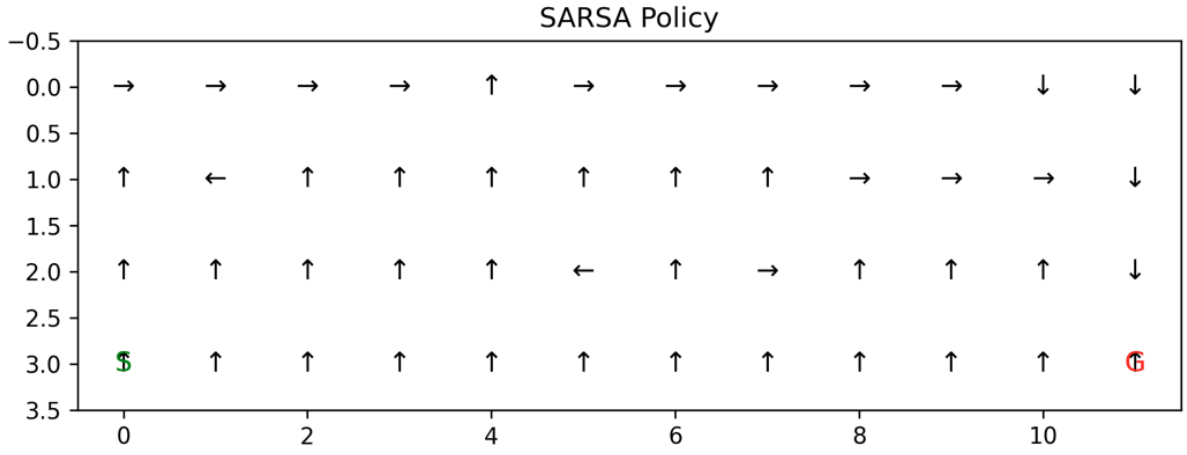
The Q-learning heatmap shows very high values along the cliff edge, revealing that the agent thinks that route is best due to being the shortest path. SARSA's values (from earlier output) are shifted upward, away from the cliff. This matches the expected behavior from theory: SARSA accounts for exploratory risk, while Q-learning does not.

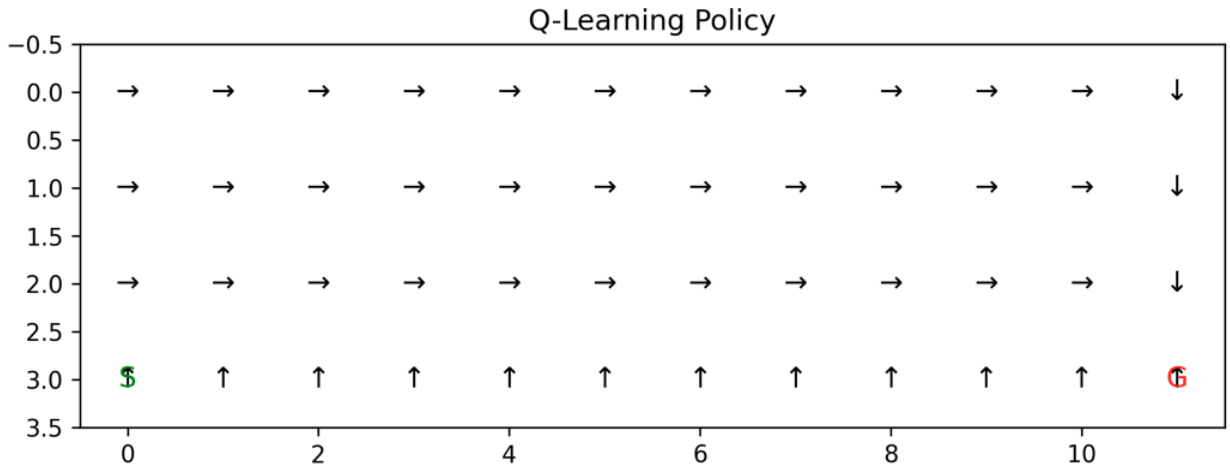





### 3. Policy Arrow Plots

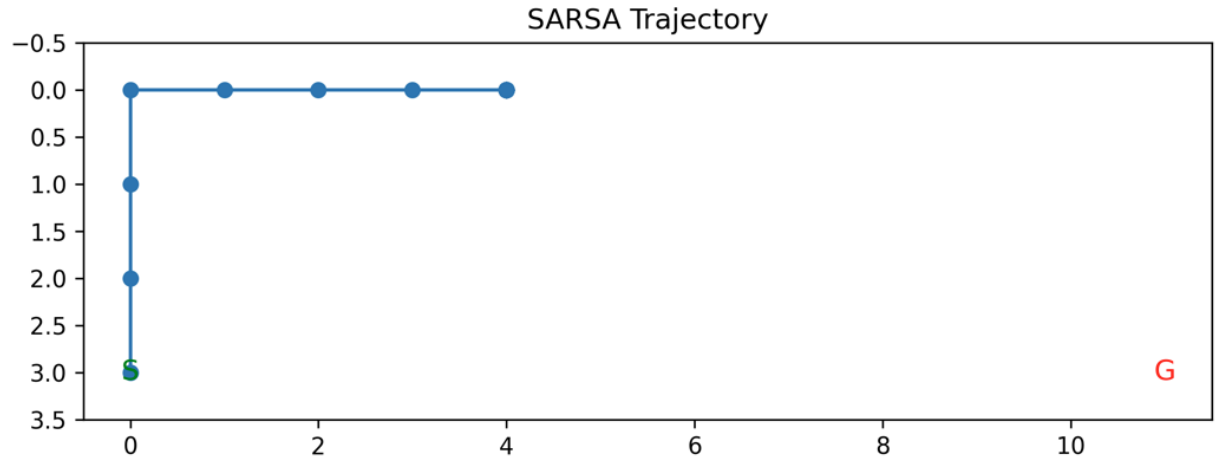
The policy arrow plots highlight how differently the two algorithms behave after training. SARSA's policy directs the agent along a safer upper route, keeping distance from the cliff, while Q-learning's policy hugs the cliff because it identifies the shortest path as optimal. This visual difference reflects the core distinction between on-policy and off-policy TD control: SARSA learns based on the actions it actually takes under ε-greedy exploration, while Q-learning learns as if it always follows the greedy action.

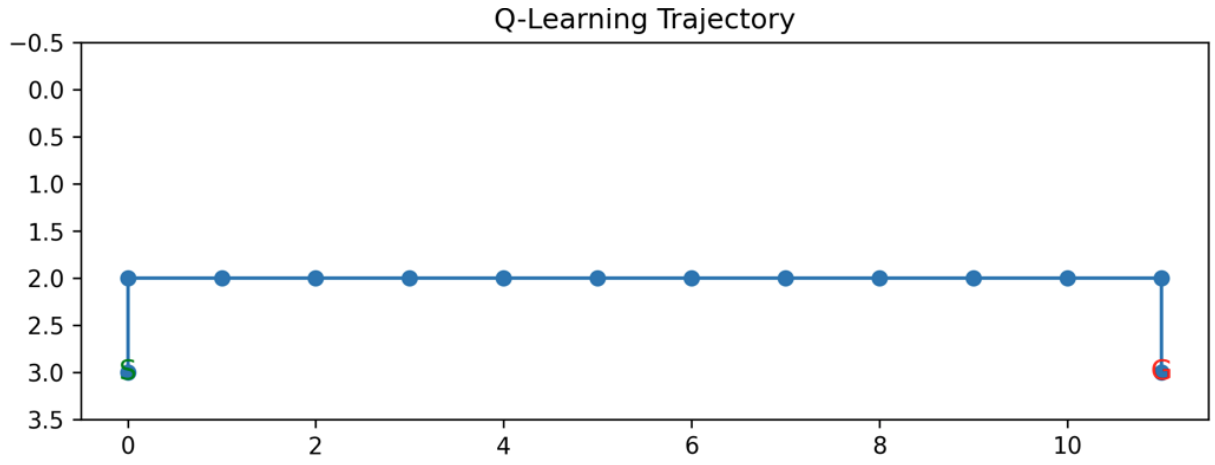





**4. Trajectory Plots**

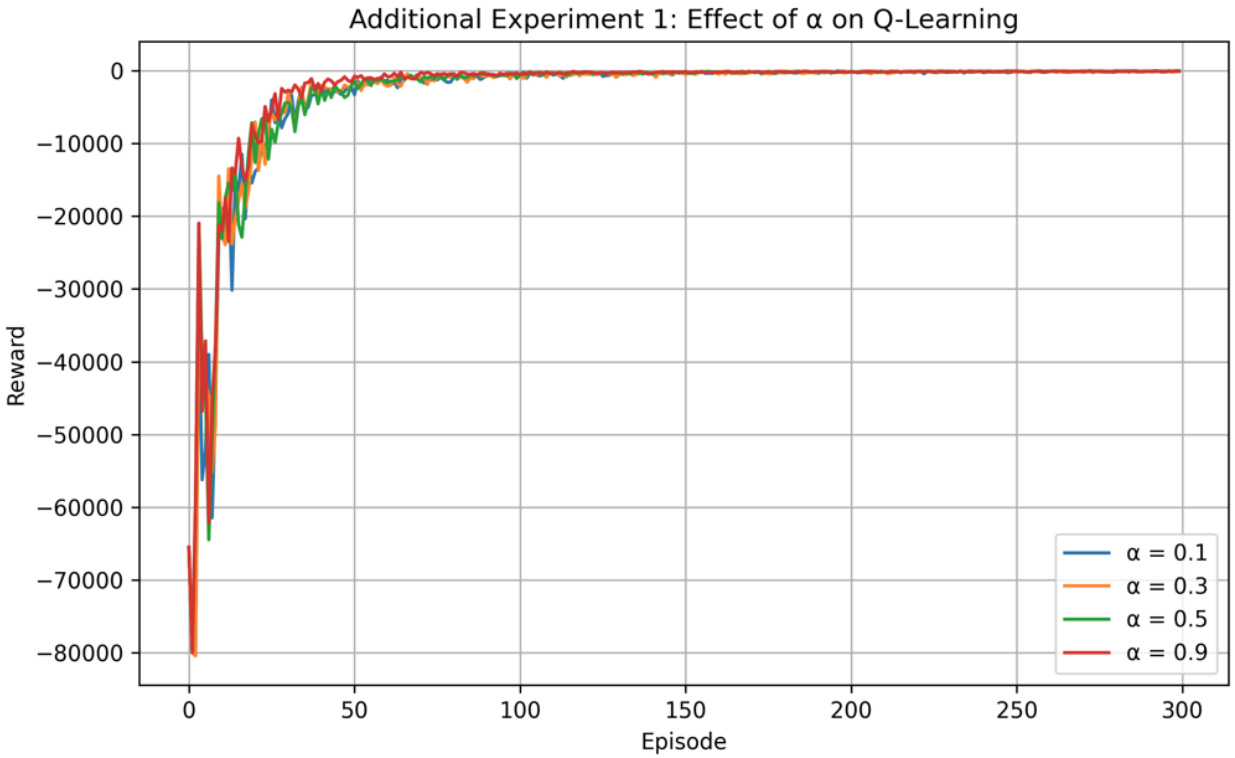
The trajectory plots clearly show the behavioral difference between the two algorithms: the SARSA agent immediately moves upward and follows a safer route away from the cliff, while the Q-learning agent travels straight along the cliff edge. These contrasting paths visually demonstrate the effect of on-policy versus off-policy updates—SARSA accounts for the risk introduced by exploration, whereas Q-learning learns the shortest but riskiest route.

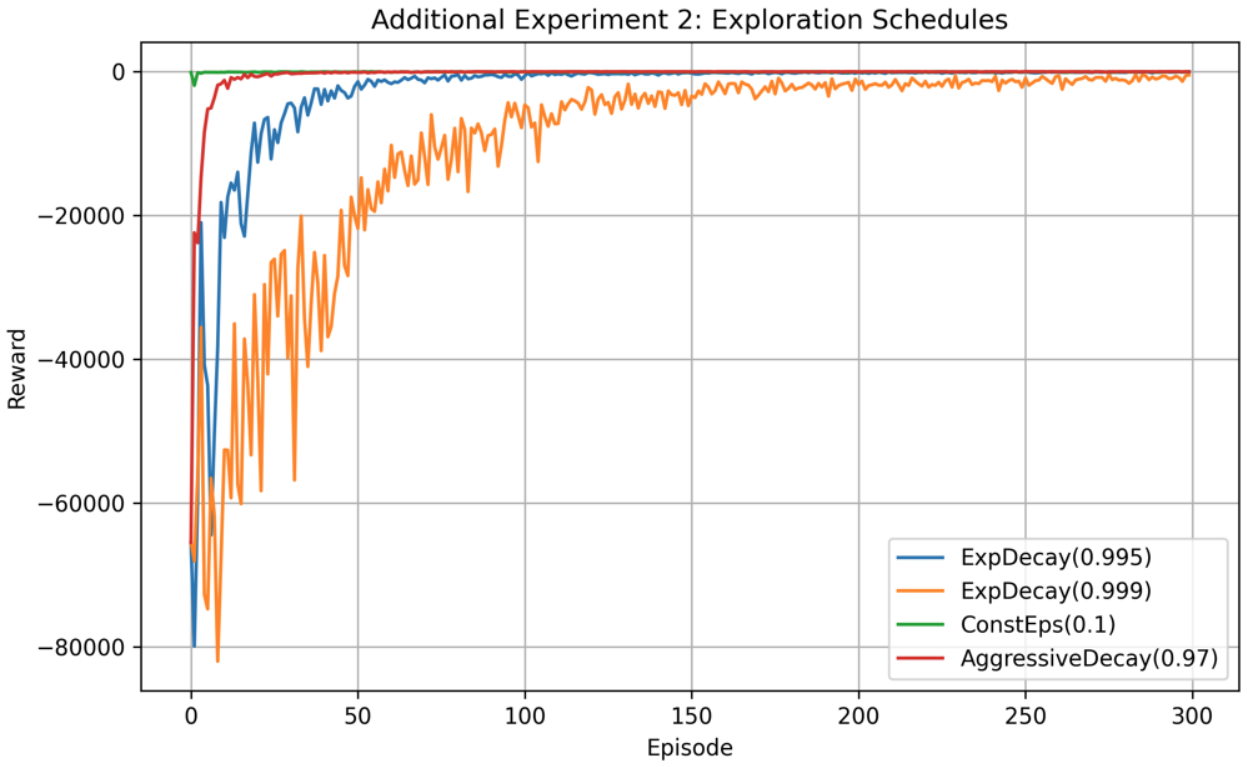


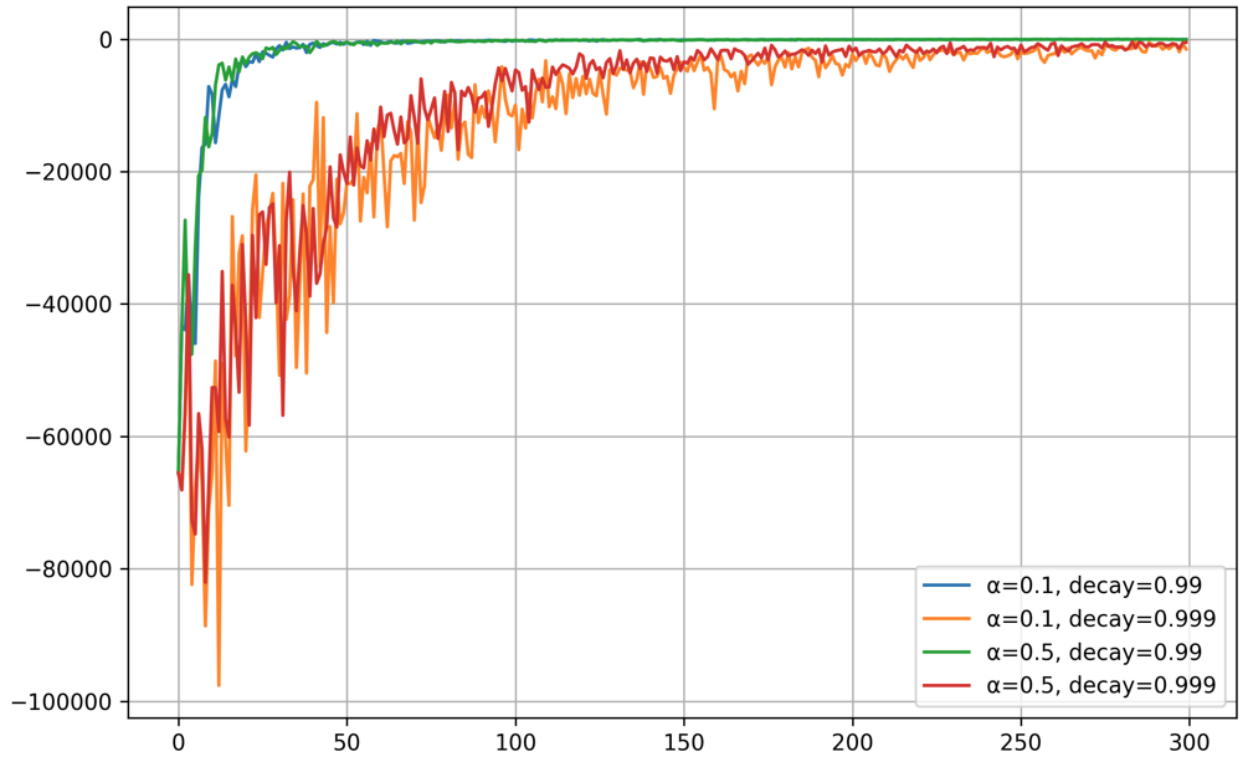



**5. Hyperparameter Experiments**

The hyperparameter experiments showed that a high learning rate (α=0.9) caused unstable early updates, while smaller α values were smoother but slower, with α=0.5 offering the best balance. Exploration schedules also made a clear difference: aggressive ε-decay (0.97) improved very quickly, constant ε=0.1 produced steady learning after a short warm-up, and slow decay (0.999) learned the slowest because it explored for too long. Overall, the combination of α=0.5 and ε-decay=0.99 provided the most stable and efficient learning performance.

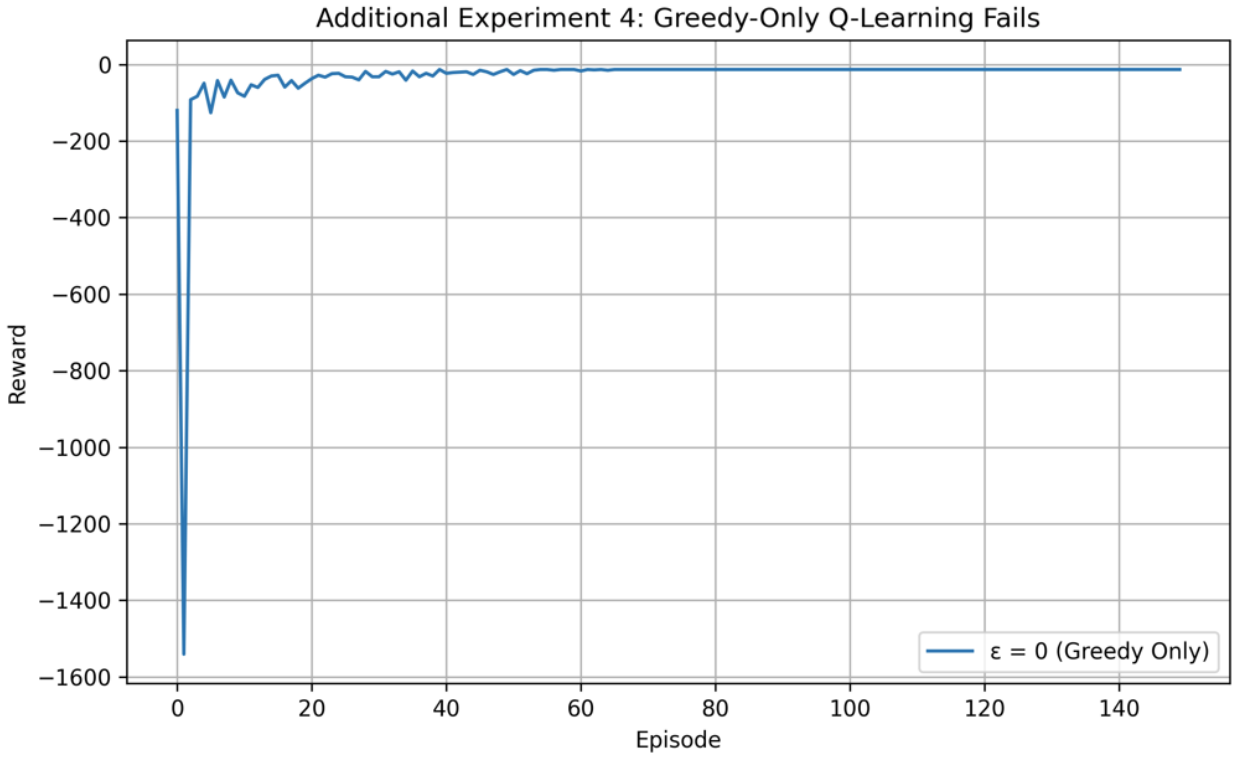





**6. Greedy-Only Q-learning (ε=0)**

The greedy-only run performed extremely poorly early on, with rewards dropping to around −1500. Without exploration, the agent fell off the cliff repeatedly and failed to discover safe alternatives. This highlights the importance of exploration for TD learning.




**Section 3: AI Use Reflection**

I started this lab by asking AI to help generate a clean structure for SARSA and Q-learning with TD(0) updates, multi-seed evaluation, and plotting. The initial code looked complete, but it failed when I ran it.

The first error came from unpacking observations. The AI-generated code was using the old Gym API and expected obs = env.reset(), but CliffWalking returns (obs, info). I shared the exact error message with AI, and it helped me update all reset() and step() calls to match the correct Gymnasium format. After fixing this, the code ran but produced blank figures.

The plots were saving as empty images. I asked AI why my saved figures were blank. It pointed out that some plots were being created without plt.show() or the figure context was overridden before saving. I inserted explicit plt.figure() calls, added plt.show(), and the issue was resolved.

When running multiple seeds, I found NaN values appearing in the Q-table. I asked AI for help again, sharing the exact printout. It explained that NaNs usually occur when indexing

is incorrect. It turned out I was treating observation values as tuples in a few places. Converting observations using int(obs) fixed the problem completely.

Throughout these iterations, I didn't accept every suggestion. For example, AI once recommended clipping Q-values, which would have masked underlying issues. I ignored that suggestion and focused on fixing indexing errors. I also manually tested a few episodes to confirm TD updates were correct.

Overall, the debugging process helped me understand how TD targets are formed, how off-policy updates depend on the max operator, and how exploration affects stability. Working with AI made the process faster, but the real learning came from analyzing mistakes and understanding why each fix mattered.

**Section 4: Speaker Notes**

- Problem: Compare SARSA (on-policy) and Q-learning (off-policy) using TD(0) in the CliffWalking environment.
- Method: Implemented both algorithms with ε-greedy exploration, online TD updates, and 30-seed averaging.
- Key Design Choice: Used ε-decay to balance exploration and stability during training.
- Main Result: SARSA learned a safe upper route, while Q-learning chose the risky cliff-edge path.
- Insight: On-policy updates internalize exploration risk; off-policy updates ignore it.
- Challenge: Debugging reset/step API issues and blank figure outputs required multiple iterations.
- Takeaway: Exploration strategy and update type strongly influence behavior in TD control.