

Learning to Navigate Dual-Reward Cliff-Walks

COM3240: Reinforcement Learning Coursework

Nirmal Eswar Vee
University of Sheffield

April 21, 2025

Abstract

This report details the process taken to solve the problem of enabling a robot to learn to navigate its surroundings and reach its target with fewer steps each episode. The State–Action–Reward–State–Action (SARSA) algorithm was used to implement the solution, as it provides an on-policy, risk-sensitive update that incorporates exploratory actions directly. This was achieved by using a decaying ϵ -greedy policy-guided exploration. Experiments show that the agent converges to the optimal high-value artefact path while minimising cliff penalties and episode length after tuning the hyperparameters. These results confirm SARSA’s effectiveness in the hazardous grid environment.

Contents

1	Introduction	4
1.1	Context	4
1.2	Motivation	4
1.3	Structure of Report	4
2	Environment	4
3	Algorithm	5
3.1	Temporal-Difference (TD) Learning	5
3.2	Q-Learning	5
3.3	SARSA	6
3.4	Why SARSA instead of Q-learning?	6
4	Experimental Setup	7
4.1	Hyperparameter Selection	7
4.2	Influence of Each Hyperparameter and Tuning	7
5	Results (Learning Curves)	8
5.1	Raw Total Reward per Episode	8
5.2	Smoothed Total Reward per Episode	8
5.3	Raw Episode Length per Episode	9
5.4	Smoothed Episode Length per Episode	10
5.5	Final Greedy Policy	11
6	Conclusion	13
6.1	Key Insights and Implications	13
6.2	Closing Comment	13

1 Introduction

This section aims to outline the background of the project and the structure of the report.

1.1 Context

The task was to train a reinforcement learning (RL) agent in a hazardous grid-world to discover high-reward paths while avoiding penalties. The agent had to learn to trade off exploration (to find distant rewards) against the risk of falling into “cliff” cells that terminate the episode with a large negative cost.

1.2 Motivation

In the specific environment that was set, two artefacts of different value were placed. One was a safer, nearby reward of +250, whilst the other was a more distant, high-value reward of +5000 across a cliff region. This dual-reward setup created a local-optimum trap since an agent that doesn’t explore enough would quickly favour the smaller artefact and never discover the truly optimal path.

1.3 Structure of Report

- The setup of the environment is described in section 2.
- I implemented an on-policy TD algorithm SARSA, which incorporated exploratory actions into its update, helping the agent learn risk-aware routes. This is discussed in section 3.
- I used different hyperparameters to balance initial search with eventual exploitation. This is discussed in section 4.
- I present learning curves (reward and episode length) and a final policy-arrow map that visualises the safe, optimal trajectory in section 5.
- I reflect on the project as a whole and share my conclusions in section 6.

2 Environment

This section describes the environment constructed in which the agent acts and the behaviour associated with the specific components in the environment.

- **Grid dimensions & obstacles:** A **13 × 10** lattice with a start cell at (11, 1). Two impassable walls occupy (7, 7)–(7, 9) and (9, 0)–(9, 3).
- **Cliffs:** “Cliff” cells (blue in Fig. 1) span the top row and two lower patches; stepping into any incurs a -100 penalty and terminates the episode.
- **Artefacts (goals):** A nearby artefact at (2, 1) yields +250, and a distant artefact at (9, 8) yields +5000; reaching either ends the episode.
- **Rebound & bounds:** If the agent moves off-grid or into walls, the agent rebounds to its previous cell; non-terminal moves otherwise incur zero reward.
- **Episode limit:** A maximum of 500 steps prevents infinite wandering.
- A visual representation of the grid can be seen in figure 1

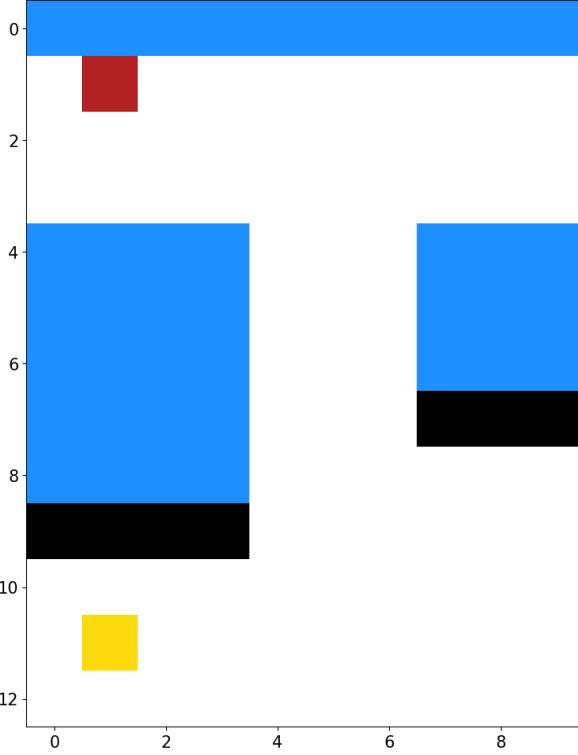


Figure 1: The environment for the agent to operate in

3 Algorithm

This section explains the fundamental maths behind TD learning algorithms and the rationale behind why SARSA was chosen for the specific implementation.

3.1 Temporal-Difference (TD) Learning

TD learning updates the value estimates immediately after each step by comparing the old estimate to a new one that includes the received reward and the next state's value [3, 1]. For a value function \mathbf{V} , the one-step update at state s_t is

$$\mathbf{V}(s_t) \leftarrow \mathbf{V}(s_t) + \alpha[r_{t+1} + \gamma \mathbf{V}(s_{t+1}) - \mathbf{V}(s_t)],$$

Where:

- α is the learning rate.
- γ the discount factor.
- δ_t is the TD error.

3.2 Q-Learning

As an off-policy algorithm, Q-Learning learns the optimal action-value function \mathbf{Q}^* independently of the agent's exploration policy [4, 1].

$$\mathbf{Q}(s_t, a_t) \leftarrow \mathbf{Q}(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_{a'} \mathbf{Q}(s_{t+1}, a') - \mathbf{Q}(s_t, a_t)].$$

Where:

- $\mathbf{Q}(s, a)$ is the action-value function for state s and action a .
- s_t is the state at time t .
- a_t is the action taken in state s_t .
- r_{t+1} is the reward received after action a_t .
- s_{t+1} is the next state.
- $\max_{a'} \mathbf{Q}(s_{t+1}, a')$ is the estimate of the best possible future value at s_{t+1} .
- α is the learning rate.
- γ is the discount factor.

The algorithm converges to the true optimum \mathbf{Q}^* under standard stochastic-approximation conditions. However, it may also overestimate values in high-penalty regions because the max-backup ignores exploratory actions.

3.3 SARSA

SARSA was first introduced as an on-policy TD control algorithm in [2]. It adjusts its Q-values by taking a single gradient step on the one-step TD error under the agent's own policy. We can think of this as minimising the squared difference

$$J(\mathbf{Q}) = \frac{1}{2} \mathbb{E}_\pi \left[(r_{t+1} + \gamma \mathbf{Q}(s_{t+1}, a_{t+1}) - \mathbf{Q}(s_t, a_t))^2 \right].$$

Where:

- $J(\mathbf{Q})$ is the mean-squared one-step TD error for the action-value function \mathbf{Q} .
- $\mathbb{E}_\pi[\cdot]$ denotes the expectation over state-action transitions when the agent acts according to policy π .

A single stochastic gradient update on $\mathbf{Q}(s_t, a_t)$ gives

$$\mathbf{Q}(s_t, a_t) \leftarrow \mathbf{Q}(s_t, a_t) + \underbrace{\alpha [r_{t+1} + \gamma \mathbf{Q}(s_{t+1}, a_{t+1}) - \mathbf{Q}(s_t, a_t)]}_{\delta_t}.$$

Because SARSA uses the same ϵ -greedy policy to pick a_{t+1} and to compute the update, it naturally accounts for exploration.

3.4 Why SARSA instead of Q-learning?

I picked SARSA because it better matches how the agent actually behaves and leads to the agent staying safer around the cliffs:

- **On-policy = real behaviour:** SARSA uses the exact same ϵ -greedy action you take next when it updates \mathbf{Q} , so its estimates reflect your real exploration rather than an idealised “always-greedy” move.
- **Risk-aware learning around cliffs:** SARSA’s update uses the actual next action a_{t+1} drawn from the ϵ -greedy policy. That policy occasionally “slips” by choosing a move adjacent to the cliff, incurring the -100 fall penalty. Because SARSA backs up these real exploratory falls into its Q-values, it learns to favour safer, slightly longer detours, whereas Q-learning’s max-backup ignores those slips and ends up hugging the cliff edge.

4 Experimental Setup

This section describes how I conducted my experiments, including hyperparameter choices, the run throughs, and the metrics I report.

- **Hyperparameters:** learning rate α , discount factor γ , initial exploration ε_0 , and exponential decay rate κ .
- **Training runs:** I evaluated each configuration over 20 independent seeds, each for 500 episodes, to account for stochasticity.
- **Metrics:** for each episode I record the total (undiscounted) reward and the number of steps to termination. After that, I then report the mean and standard deviation across the 20 seeds at each episode.

4.1 Hyperparameter Selection

I tested each parameter over the following candidate sets:

$$\alpha \in \{0.05, 0.10, 0.20\}, \quad \gamma \in \{0.90, 0.99\}, \quad \varepsilon_0 \in \{0.50, 1.00\}, \quad \kappa \in \{0.005, 0.01, 0.02\}.$$

For each combination I ran 20 independent seeds and plotted the per-episode mean \pm 1 std learning curve. Table 1 summarises the original “Baseline” defaults and the “Tuned” configuration that had the fastest convergence in those averaged results.

4.2 Influence of Each Hyperparameter and Tuning

- α (learning rate): Larger α (e.g. 0.20) speeds up early updates but increases variance, whilst smaller α (e.g. 0.05) has smoother but slower convergence.
- γ (discount factor): $\gamma = 0.99$ places more weight on the distant +5000 reward, whereas $\gamma = 0.90$ biases the agent toward the nearer +250 artefact and proved to converge faster under my policy and decay schedule.
- ε_0 (initial exploration): A high initial exploration rate ($\varepsilon_0 = 1.00$) allowed reliable discovery of the far artefact, lowering to $\varepsilon_0 = 0.50$ in the tuned run still discovered the artefact quickly while reducing early-episode variance.
- κ (decay rate): A moderate decay ($\kappa = 0.01$) balanced exploration and exploitation in the baseline, whereas a slightly faster decay ($\kappa = 0.02$) in the tuned run concentrated learning on the +250 artefact before settling on the optimal path.

Param	α	γ	ε_0	κ
Baseline	0.10	0.99	1.00	0.01
Tuned (Best found)	0.10	0.90	0.50	0.02

Table 1: Hyperparameter settings: “Baseline” are the course-default values, and “Tuned” are the parameters that achieved the fastest convergence in the mean \pm std learning curves (20-seed average).

5 Results (Learning Curves)

In this section I visualise the agent's progress through reward and step-length plots. For each metric, I show the tuned (best-found) configuration first and then the baseline configuration for comparison.

5.1 Raw Total Reward per Episode

The raw per-episode total reward curves (mean \pm 1 std, clipped to $[-100, 5000]$) show how quickly the agent discovers the distant +5000 artefact under each configuration.

Tuned (Best-found) run:

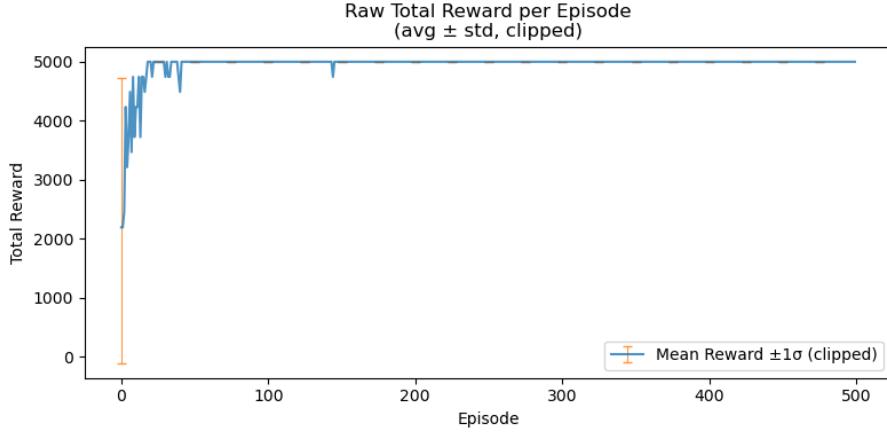


Figure 2: Raw total reward per episode (mean \pm std, clipped), averaged over 20 seeds, using the tuned parameters $\{\alpha = 0.10, \gamma = 0.90, \varepsilon_0 = 0.50, \kappa = 0.02\}$.

Baseline run:

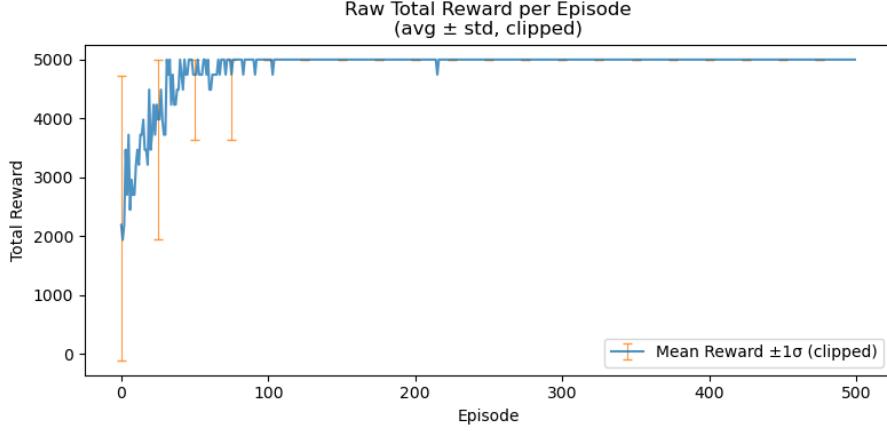


Figure 3: Raw total reward per episode (mean \pm std, clipped), averaged over 20 seeds, using the baseline parameters $\{\alpha = 0.10, \gamma = 0.99, \varepsilon_0 = 1.00, \kappa = 0.01\}$.

Key difference: The tuned configuration (Fig. 2) reaches the +5000 reward reliably within the first 50 episodes, whereas the baseline (Fig. 3) takes around 100 episodes on average to achieve the same.

5.2 Smoothed Total Reward per Episode

Applying a 20-episode moving average filters noise and shows convergence trends.

Tuned (Best-found) run:

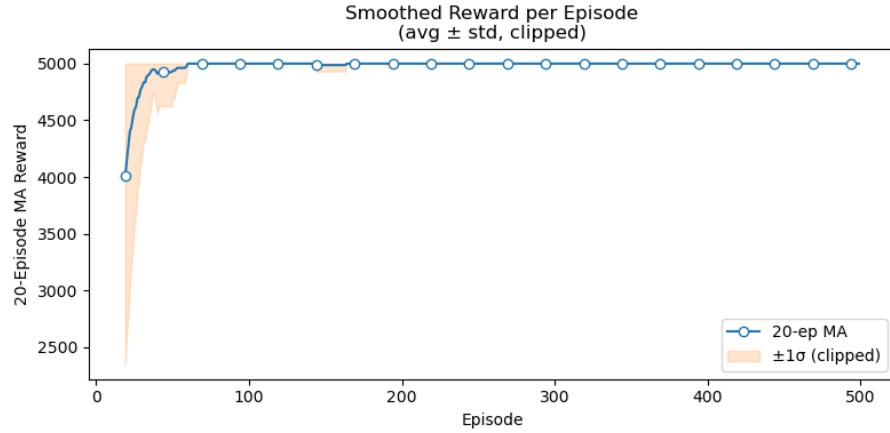


Figure 4: 20-episode moving average of total reward (mean \pm std, clipped), averaged over 20 seeds, using the tuned parameters.

Baseline run:

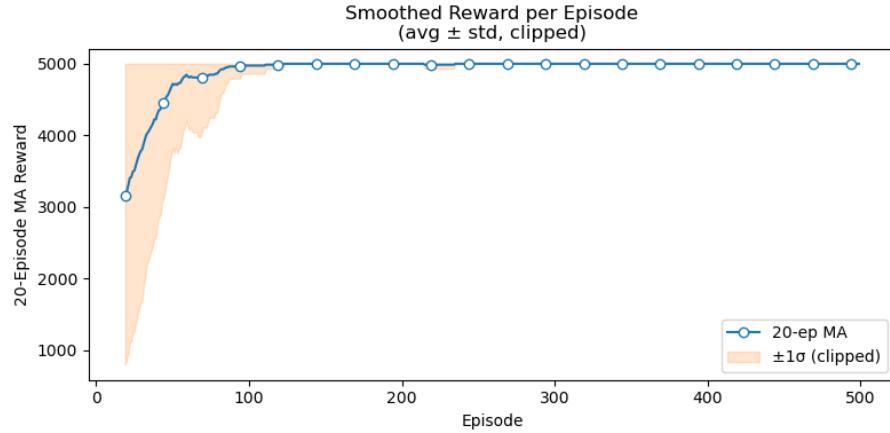


Figure 5: 20-episode moving average of total reward (mean \pm std, clipped), averaged over 20 seeds, using the baseline parameters.

Key difference: The tuned run plateaus near +5000 by episode 40 (Fig. 4), while the baseline only approaches +5000 around episode 100 (Fig. 5).

5.3 Raw Episode Length per Episode

Raw episode length shows how many steps the agent takes before termination each episode.

Tuned (Best-found) run:

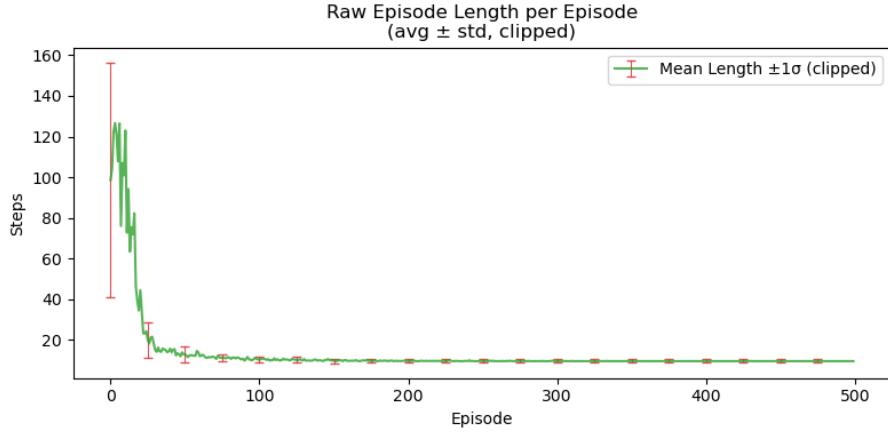


Figure 6: Raw episode length (mean \pm std, clipped), averaged over 20 seeds, using the tuned parameters.

Baseline run:

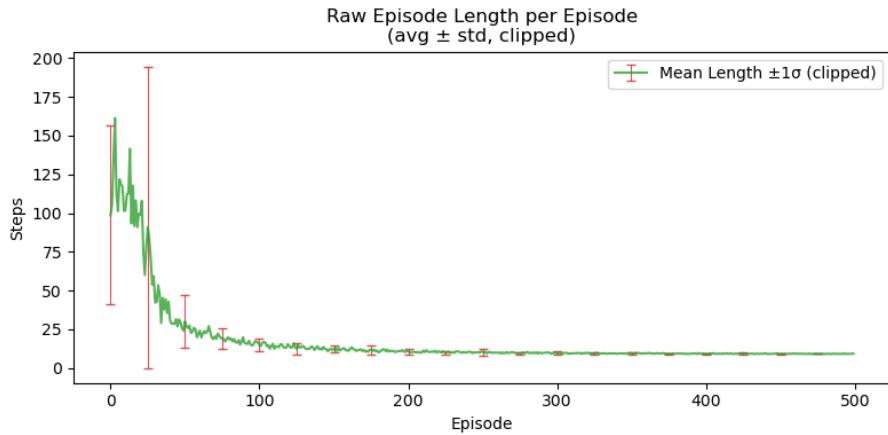


Figure 7: Raw episode length (mean \pm std, clipped), averaged over 20 seeds, using the baseline parameters.

Key difference: Under the tuned settings, episode length drops below 20 steps by episode 60 (Fig. 6); the baseline only settles below 20 steps around episode 150 (Fig. 7).

5.4 Smoothed Episode Length per Episode

A 20-episode moving average on episode length filters out exploration spikes.

Tuned (Best-found) run:

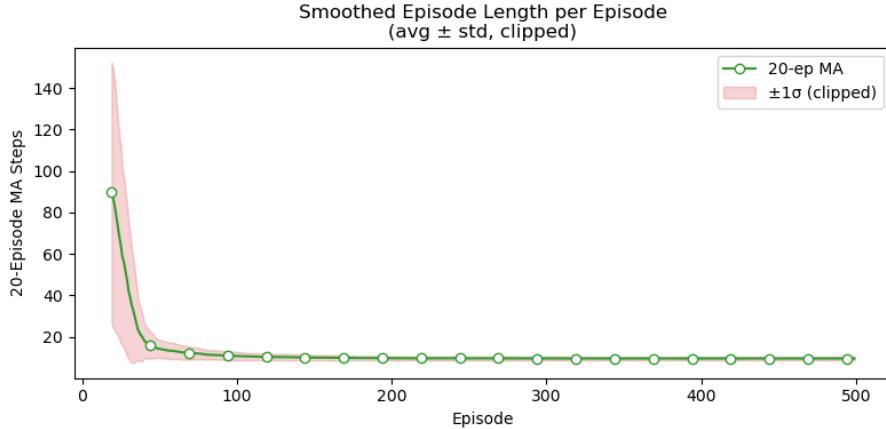


Figure 8: 20-episode moving average of episode length (mean \pm std, clipped), averaged over 20 seeds, using the tuned parameters.

Baseline run:

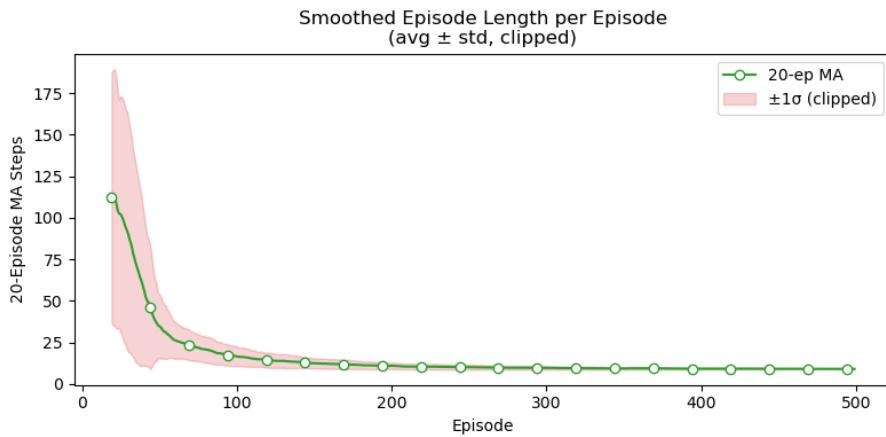


Figure 9: 20-episode moving average of episode length (mean \pm std, clipped), averaged over 20 seeds, using the baseline parameters.

Key difference: The tuned policy reaches sub-10 steps by episode 60 (Fig. 8), whereas the baseline only dips below 10 steps near episode 150 (Fig. 9).

5.5 Final Greedy Policy

The arrow-map visualises the learned greedy policy over the grid.

Tuned (Best-found) run:

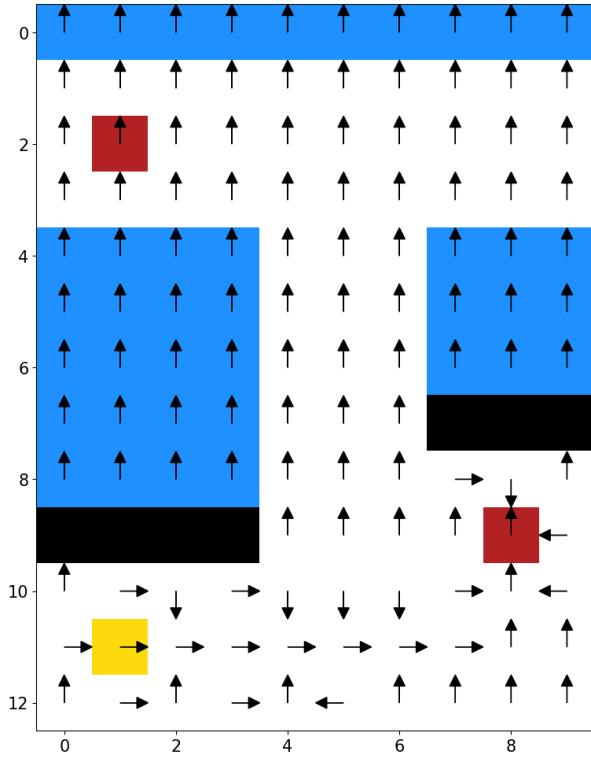


Figure 10: Final greedy policy under tuned parameters: arrows point to the next-step action, showing a clear detour around the cliff to the distant artefact.

Baseline run:

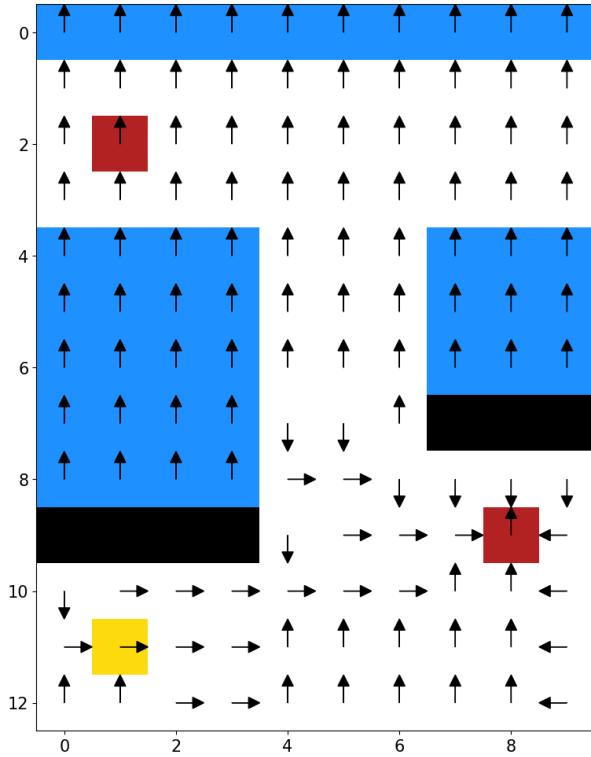


Figure 11: Final greedy policy under baseline parameters: occasional arrows point into cliff regions, reflecting higher risk in the learned policy.

Key difference: The tuned policy (Fig. 10) cleanly skirts the cliff edge, whereas the baseline (Fig. 11) retains some cliff-pointing actions from early exploration.

6 Conclusion

I evaluated both a conservative baseline configuration ($\{\alpha = 0.10, \gamma = 0.99, \varepsilon_0 = 1.00, \kappa = 0.01\}$) and a tuned configuration ($\{\alpha = 0.10, \gamma = 0.90, \varepsilon_0 = 0.50, \kappa = 0.02\}$) over 20 independent seeds, reporting the per-episode mean and standard deviation. Averaging across multiple runs confirmed that the tuned parameters converge substantially faster (reaching the +5000 artefact in under 60 episodes on average) than the baseline (which requires around 100–150 episodes).

By using on-policy SARSA with an exponentially decaying ε -greedy schedule, my agent initially explored broadly, then steadily exploited the high-value path. The result is a risk-aware policy that reliably avoids the cliff and secures the +5000 reward in minimal steps.

6.1 Key Insights and Implications

- **Averaging reduces variance:** Reporting mean \pm std over 20 seeds provides a clearer picture of true performance and avoids over-fitting to a fluke run.
- **Parameter sensitivity:** Lowering γ to 0.90 and decreasing initial exploration to $\varepsilon_0 = 0.50$ (with a slightly faster decay) made convergence happen quicker by focusing learning on the +250 artefact first, then committing to the +5000 artefact once discovered.
- **Risk-aware on-policy control:** SARSA's on-policy update naturally accounts for "slips", producing safer detours around hazards compared to off-policy methods.

6.2 Closing Comment

Overall, SARSA with a carefully tuned decaying exploration schedule achieves both fast convergence and low cliff-fall risk in this dual-reward cliff-walk environment. Future work could compare these findings to other exploration strategies (e.g. Softmax) or extend to larger domains.

References

- [1] Kaelbling, L. P., Littman, M. L., and Moore, A. W. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4 (1996), 237–285.
- [2] Rummery, G. A., and Niranjan, M. Line-learning algorithms for temporal difference learning. In *Machine Learning: Proceedings of the 1994 UK Workshop* (1994), pp. 230–234.
- [3] Sutton, R. S., and Barto, A. G. *Reinforcement Learning: An Introduction*. MIT Press, 2018.
- [4] Watkins, C. J., and Dayan, P. Q-learning. *Machine Learning* 8, 3-4 (1992), 279–292.

Appendix

AI Assistance in Formatting and Grammar

Throughout the preparation of this paper, I made use of generative AI to assist with formatting and grammatical precision. Specifically, I used AI to:

- Ensure consistent formatting of mathematical equations and textual elements.
- Enhance the grammar accuracy of some sentences.
- Verify that technical terms and symbols were used consistently and correctly.

AI Assistance in Finding Specific References

In addition to formatting and grammar support, AI was instrumental in helping locate and verify specific references relevant to the topics discussed in this paper. This involved:

- Utilising AIs to help find specific academic papers, articles, and publications.
- Ensuring that all references were correctly formatted according to the required citation style.

All ideas and information gathered were my own.