AE 598 HW2- Implementation of three Model-Free RL Algorithms for Frozen Lake in a Tabular Setting

Nitya Jagadam*
University of Illinois Urbana-Champaign, IL, 61820, USA

This report presents the implementation and evaluation of three fundamental model-free reinforcement learning algorithms: On-Policy First-Visit Monte Carlo (MC) Control, SARSA, and Q-Learning within a tabular setting using the *FrozenLake-v1* environment from the Gymnasium library. The environment provides a discrete, fully observable Markov Decision Process (MDP) that enables comparison between on-policy and off-policy learning under both deterministic (`is_slippery=False`) and stochastic (`is_slippery=True`) dynamics. Each algorithm learns an action-value function $Q(s, a)$ to derive an optimal policy through ϵ-greedy exploration. The performance of the agents is analyzed through evaluation return versus time steps, along with visualizations of the resulting value functions and policies. Experimental results show that Q-Learning achieves the fastest convergence in deterministic settings, while SARSA exhibits greater robustness under stochastic transitions. These findings highlight the trade-offs between policy stability and learning efficiency across different tabular control methods.

I. Introduction

Reinforcement Learning (RL) provides a computational framework for an agent to learn optimal behaviors through direct interaction with an environment, guided only by scalar reward feedback. Formally, RL problems are modeled as Markov Decision Processes (MDPs), defined by a set of states, actions, transition probabilities, and rewards. The agent's objective is to learn a policy $\pi(a|s)$ that maximizes the expected discounted return

$$G_t = \mathbb{E}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}\right],$$

where $\gamma \in [0, 1)$ is the discount factor controlling the weighting of future rewards.

In many realistic environments, the transition and reward models are unknown, making *model-free* methods crucial. Such algorithms learn directly from sampled experience without explicitly constructing the environment model. Within this class, tabular methods are the most fundamental approach, where the state and action spaces are small enough for each state-action pair to be explicitly represented and updated in a lookup table. Despite their simplicity, tabular methods provide the foundation for understanding and extending to function approximation and deep reinforcement learning.

This homework focuses on three cornerstone model-free control algorithms that differ in how they estimate the action-value function $Q(s, a)$ and update the policy:

1) **On-Policy First-Visit Monte Carlo (MC) Control** – learns from complete episodes by averaging returns from first visits to each (s, a) pair.
2) **SARSA (State–Action–Reward–State–Action)** – an on-policy Temporal Difference (TD) method that updates estimates based on the action actually taken by the policy.
3) **Q-Learning** – an off-policy TD method that updates towards the greedy action, independent of the agent's current policy.

To study and compare these algorithms, we employ the *FrozenLake-v1* environment provided by the Gymnasium library [1]. This environment defines a small, discrete MDP where the agent must navigate a 4×4 grid of frozen tiles to reach a goal while avoiding holes. Each episode begins at the start state ('S') and terminates when the agent either falls into a hole ('H') or reaches the goal ('G'). Intermediate states ('F') are safe frozen surfaces. The state space thus contains 16 discrete positions, and the action space includes four possible moves: *LEFT*, *DOWN*, *RIGHT*, and *UP*. The reward structure is sparse: a reward of +1 is given upon reaching the goal, and 0 otherwise.

*UIN: 659343170, AE 598: Reinforcement Learning, Fall 2025.

Two versions of the environment are evaluated:

- `is_slippery=False`: Deterministic transitions where actions yield predictable outcomes.
- `is_slippery=True`: Stochastic transitions simulating the agent slipping, introducing uncertainty into the outcome of each move.

By comparing performance across these two dynamics, this study examines how on-policy and off-policy learning strategies handle exploration, stability, and convergence under uncertainty. The learning behavior of each algorithm is evaluated using two key diagnostics:

1) The evaluation return plotted versus the number of time steps, showing the rate and stability of learning.
2) The final policy and value function visualizations, revealing the learned behavior and value distribution across the grid.

Overall, this assignment provides a hands-on understanding of the core mechanisms underlying policy evaluation and control in reinforcement learning. The comparison between Monte Carlo, SARSA, and Q-Learning illustrates the trade-offs between bias, variance, sample efficiency, and robustness—concepts that form the foundation for more advanced RL algorithms and applications.

II. Methods

This section describes the formulation of the FrozenLake environment as a Markov Decision Process (MDP), followed by detailed explanations of the three tabular model-free control algorithms implemented in this study: Monte Carlo Control, SARSA, and Q-Learning.

A. Markov Decision Process Definition

The FrozenLake problem can be represented as a finite MDP defined by the tuple

$$\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$$

where:

- **State space** $\mathcal{S} = \{0, 1, 2, \ldots, 15\}$ corresponds to the 16 discrete grid positions on the 4×4 map.
- **Action space** $\mathcal{A} = \{0 : \text{LEFT}, 1 : \text{DOWN}, 2 : \text{RIGHT}, 3 : \text{UP}\}$.
- **Transition model** $P(s'|s, a)$ defines the probability of moving to state s' from s after taking action a. When `is_slippery=False`, transitions are deterministic; when `is_slippery=True`, stochastic outcomes simulate slipping.
- **Reward function** $R(s, a, s') = 1$ when s' is the goal state, and 0 otherwise.
- **Discount factor** $\gamma = 0.95$.

Episodes terminate upon reaching a hole or the goal, and the objective is to learn an optimal policy π^* that maximizes expected return.

B. Algorithm Summaries

All three algorithms maintain an action-value function $Q(s, a)$ stored in a tabular form. Learning proceeds through repeated interactions with the environment, using ϵ-greedy exploration:

$$a_t = \begin{cases} \text{random action}, & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a), & \text{with probability } 1 - \epsilon \end{cases}$$

After each episode or step, the Q-table is updated according to the specific algorithm.

1. On-Policy First-Visit Monte Carlo (MC) Control

The Monte Carlo control algorithm estimates the action-value function $Q(s, a)$ by averaging the empirical returns following first visits to each state–action pair within complete episodes [2]. Because it requires full episode rollouts before updates, it does not bootstrap and thus has high variance but low bias. The policy is improved iteratively using an ϵ-greedy strategy to balance exploration and exploitation. Although Monte Carlo learning converges slowly, it provides an unbiased estimate of expected returns and serves as a useful baseline for evaluating temporal-difference methods.

On-policy first-visit MC control (for ε-soft policies), estimates $\pi \approx \pi_*$

Algorithm parameter: small $\varepsilon > 0$
Initialize:
  $\pi \leftarrow$ an arbitrary ε-soft policy
  $Q(s,a) \in \mathbb{R}$ (arbitrarily), for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$
  $Returns(s,a) \leftarrow$ empty list, for all $s \in \mathcal{S}$, $a \in \mathcal{A}(s)$

Repeat forever (for each episode):
  Generate an episode following π: $S_0, A_0, R_1, \ldots, S_{T-1}, A_{T-1}, R_T$
  $G \leftarrow 0$
  Loop for each step of episode, $t = T-1, T-2, \ldots, 0$:
    $G \leftarrow \gamma G + R_{t+1}$
    Unless the pair S_t, A_t appears in $S_0, A_0, S_1, A_1 \ldots, S_{t-1}, A_{t-1}$:
      Append G to $Returns(S_t, A_t)$
      $Q(S_t, A_t) \leftarrow \text{average}(Returns(S_t, A_t))$
      $A^* \leftarrow \text{argmax}_a Q(S_t, a)$ (with ties broken arbitrarily)
      For all $a \in \mathcal{A}(S_t)$:

$$\pi(a|S_t) \leftarrow \begin{cases} 1 - \varepsilon + \varepsilon/|\mathcal{A}(S_t)| & \text{if } a = A^* \\ \varepsilon/|\mathcal{A}(S_t)| & \text{if } a \neq A^* \end{cases}$$

Fig. 1 Monte Carlo On-Policy Control Pseudocode

2. SARSA (On-Policy TD(0))

SARSA is an on-policy temporal-difference (TD) algorithm that updates the action-value function after each step by using the next action actually selected by the current policy [2]. This introduces bootstrapping and enables incremental updates without waiting for episode termination. The method captures the behavior of the current policy, leading to smoother and more stable learning in stochastic environments. However, because it follows the policy it is evaluating, convergence can be slower compared to off-policy methods such as Q-Learning.

SARSA performs temporal-difference (TD) learning by updating the Q-value after every step using the next action selected by the current policy:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t) \right]$$

This on-policy nature generally produces smoother learning curves but may converge more slowly.

Sarsa (on-policy TD control) for estimating $Q \approx q_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s,a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Choose A from S using policy derived from Q (e.g., ε-greedy)
  Loop for each step of episode:
    Take action A, observe R, S'
    Choose A' from S' using policy derived from Q (e.g., ε-greedy)
    $Q(S,A) \leftarrow Q(S,A) + \alpha\big[R + \gamma Q(S', A') - Q(S,A)\big]$
    $S \leftarrow S'$; $A \leftarrow A'$;
  until S is terminal

Fig. 2 SARSA Pseudocode

3. Q-Learning (Off-Policy TD(0))

Q-Learning is an off-policy TD control algorithm that updates the $Q(s,a)$ values toward the greedy estimate of the next state, independent of the policy being followed [2]. This allows the agent to learn the optimal action-value function while still exploring through an ϵ-greedy behavior policy. Q-Learning typically converges faster and achieves higher asymptotic performance in deterministic environments, but its reliance on maximization can make it more sensitive to noise and over-estimation in stochastic settings. z Q-Learning differs by updating toward the greedy action in the next

state, making it off-policy:

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

This often leads to faster convergence but can be more sensitive to stochasticity.

Q-learning (off-policy TD control) for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0, 1]$, small $\varepsilon > 0$
Initialize $Q(s, a)$, for all $s \in \mathcal{S}^+, a \in \mathcal{A}(s)$, arbitrarily except that $Q(terminal, \cdot) = 0$

Loop for each episode:
  Initialize S
  Loop for each step of episode:
    Choose A from S using policy derived from Q (e.g., ε-greedy)
    Take action A, observe R, S'
    $Q(S, A) \leftarrow Q(S, A) + \alpha \left[R + \gamma \max_a Q(S', a) - Q(S, A) \right]$
    $S \leftarrow S'$
  until S is terminal

Fig. 3 Q-learning Pseudocode

C. Hyperparameters

All algorithms use consistent hyperparameters to ensure fair comparison. Table 1 summarizes the key settings.

Table 1 Training Hyperparameters

Parameter	Value
Learning rate (α)	0.10
Discount factor (γ)	0.95
Exploration rate (ϵ)	0.10 (decayed to 0.01)
Episodes	30,000
Evaluation frequency	Every 256 episodes
Environment variants	`is_slippery=False, True`

D. Evaluation Procedure

After every fixed number of training episodes, a deterministic policy derived from Q was evaluated over 100 test episodes (with $\epsilon = 0$) to compute the average return. This *evaluation return* metric quantifies learning progress and stability, following the approach discussed in Albrecht *et al.* [3].

III. Results and Discussion

This section presents the learning performance and policy/value visualizations obtained from the three implemented algorithms: On-Policy First-Visit Monte Carlo (MC) Control, SARSA, and Q-Learning. Each agent was trained on both deterministic (`is_slippery=False`) and stochastic (`is_slippery=True`) variants of the FrozenLake-v1 environment. Figures 4 and 5 show the evolution of evaluation return versus environment steps, while Figures 6–8 illustrate the learned policies and corresponding state-value functions.

A. Deterministic Case (`is_slippery=False`)

In the deterministic environment, all three algorithms successfully converged to optimal goal-reaching policies. As shown in Figure 4, the evaluation return quickly rises to unity for Q-Learning, indicating fast convergence due to its off-policy maximization. Both SARSA and Monte Carlo achieve similar final performance but exhibit more transient fluctuations in early training due to on-policy exploration and episodic sampling variability, respectively.

Q-Learning's convergence is nearly monotonic, as the deterministic transition model allows its greedy updates to reflect true expected outcomes. SARSA converges slightly slower because it evaluates the policy it follows, making its updates more conservative. Monte Carlo requires complete episodes before updating, which slows learning further but leads to stable long-term convergence.

The learned policies for each method (Figures 6, 7, 8) clearly navigate around holes to reach the goal via the shortest path. The corresponding value functions highlight increasing state values toward the goal cell, confirming correct credit assignment through temporal or episodic returns.

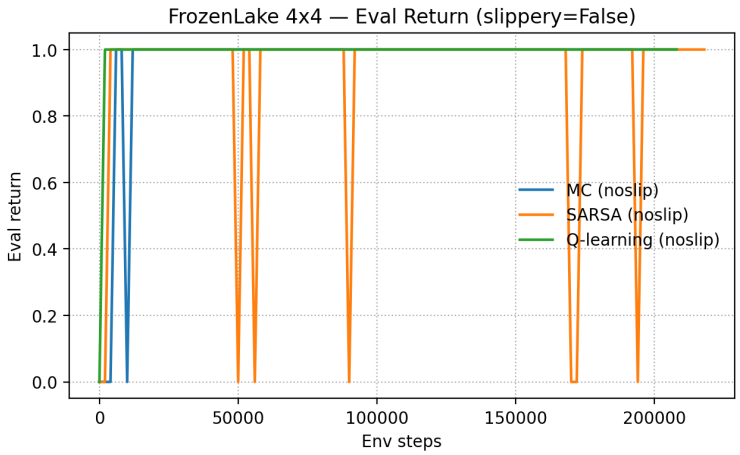

Fig. 4 Evaluation return vs. environment steps for deterministic case (`is_slippery=False`).

B. Stochastic Case (`is_slippery=True`)

In the stochastic environment, learning becomes more challenging due to non-deterministic transitions. Figure 5 shows that all three methods exhibit significant variability in returns, with Q-Learning still achieving the highest average return (approximately 0.7) but also displaying large oscillations caused by overestimation bias. SARSA demonstrates more stable learning, with smoother convergence around a moderate return level, reflecting its tendency to learn safer policies when transitions are uncertain. Monte Carlo, in contrast, converges slowly and maintains lower returns, as its episodic updates struggle to generalize under stochasticity.

The policy and value plots reveal that under slipping conditions, all algorithms learn more conservative paths. Q-Learning retains a preference for the shortest route but occasionally includes riskier actions near holes. SARSA, on the other hand, adjusts its policy to avoid high-risk transitions, resulting in lower but steadier returns. Monte Carlo's learned value map remains less sharply defined, reflecting its slower adaptation to stochastic feedback.

C. Comparative Analysis

Across both environments, Q-Learning consistently achieves the fastest convergence and highest returns, validating its efficiency in deterministic domains. However, its performance degradation under stochasticity underscores the limitations of off-policy bootstrapping when the transition dynamics are uncertain. SARSA provides a middle ground, converging slightly slower but maintaining greater robustness to noise. Monte Carlo control, though conceptually straightforward and unbiased, suffers from high variance and sample inefficiency.

These findings align closely with the theoretical expectations outlined by Sutton and Barto [2]. In particular, TD methods (SARSA and Q-Learning) outperform Monte Carlo by leveraging bootstrapping, while on-policy versus off-policy distinctions explain the stability–speed trade-off observed experimentally. Overall, the results demonstrate that while all three methods can learn effective policies in small discrete environments, Q-Learning offers the most practical balance of efficiency and performance in deterministic domains, whereas SARSA is preferable when robustness under uncertainty is critical.

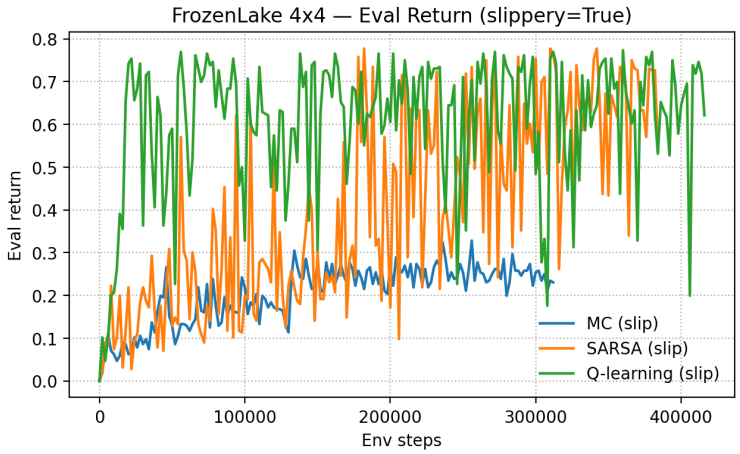

Fig. 5 Evaluation return vs. environment steps for stochastic case (`is_slippery=True`).

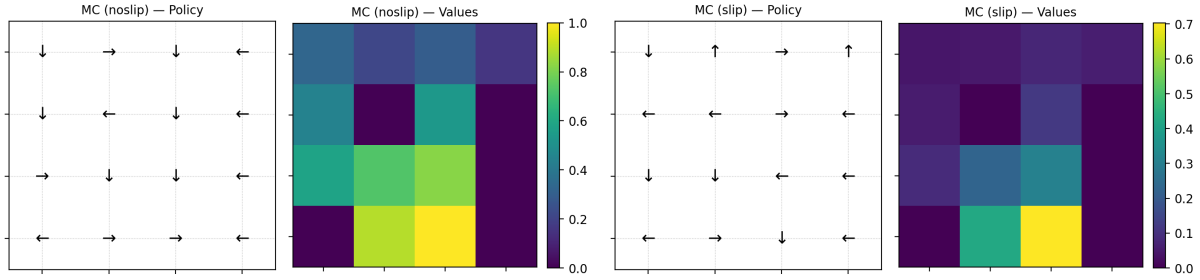

Fig. 6 Monte Carlo Control: learned policies and value functions under deterministic (top) and stochastic (bottom) conditions.

D. Discussion of Results

The learning curves in Figures 4 and 5 and the policy–value visualizations in Figures 6–8 collectively demonstrate how algorithmic structure and environmental stochasticity influence convergence, stability, and policy behavior. In the deterministic case (Figure 4), all three algorithms ultimately reach the optimal goal-reaching policy, but Q-Learning converges most rapidly and smoothly, confirming the efficiency of its off-policy updates toward the greedy action. SARSA follows closely, showing slightly slower but steadier progress due to its on-policy nature, while Monte Carlo exhibits delayed and more variable improvement because updates occur only at the end of full episodes. The corresponding value maps show sharp gradients toward the goal for Q-Learning and SARSA, indicating effective temporal credit assignment, whereas Monte Carlo's surface is flatter, consistent with higher variance in return estimates. When stochastic transitions are introduced (Figure 5), Q-Learning's performance becomes more oscillatory as overestimation bias interacts with noisy outcomes, yet it still achieves the highest average return. SARSA adapts more conservatively, yielding smoother but slightly lower returns, while Monte Carlo struggles to propagate reliable value information. The policies under slip conditions (Figures 7–8) highlight this contrast: Q-Learning retains direct but risk-prone paths, SARSA prefers safer routes with more gradual value transitions, and Monte Carlo shows diffuse, less coherent actions. Together, these results reaffirm theoretical expectations from Sutton and Barto [2]: temporal-difference methods outperform Monte Carlo in sample efficiency, Q-Learning excels in deterministic environments, and SARSA provides greater robustness under uncertainty.

IV. Conclusions

This work implemented and compared three fundamental model-free reinforcement learning algorithms—On-Policy First-Visit Monte Carlo, SARSA, and Q-Learning—on the FrozenLake-v1 environment under both deterministic and stochastic dynamics. The comparative results demonstrate that all algorithms ultimately learn effective goal-reaching

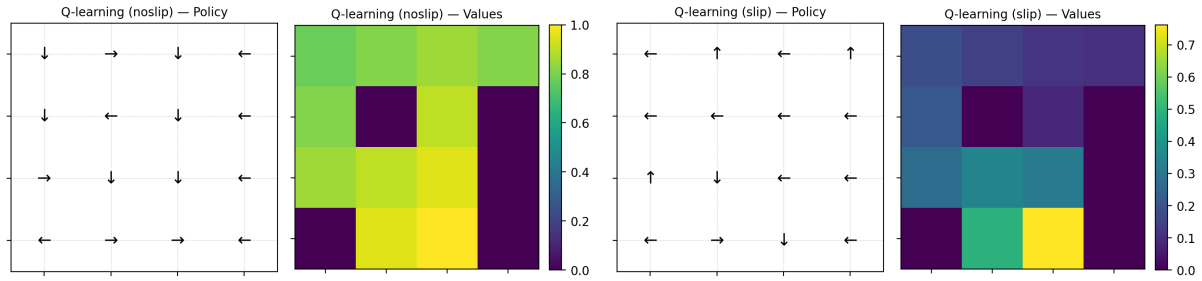

Fig. 7 Q-Learning: learned policies and value functions under deterministic (top) and stochastic (bottom) conditions.

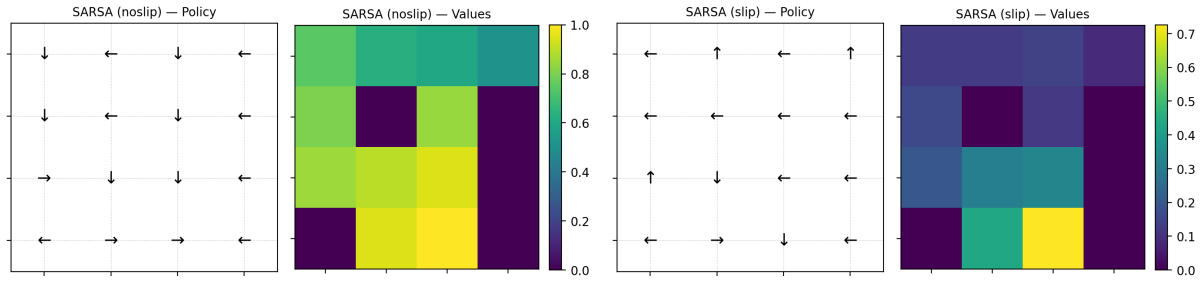

Fig. 8 SARSA: learned policies and value functions under deterministic (top) and stochastic (bottom) conditions.

policies but differ substantially in convergence speed, stability, and robustness. Q-Learning achieved the fastest and most efficient learning in deterministic settings, leveraging off-policy updates to propagate rewards aggressively across the state space. SARSA exhibited slower but smoother learning, producing more conservative policies that performed reliably under stochastic transitions. Monte Carlo control, while unbiased in principle, suffered from high variance and slower convergence due to its reliance on episodic returns. The value and policy visualizations further confirmed that temporal-difference methods propagate value information more effectively than Monte Carlo, with Q-Learning forming sharper gradients and SARSA showing greater resilience to noise. Overall, the findings align with theoretical expectations from Sutton and Barto [2], underscoring the efficiency of temporal-difference methods for tabular control and highlighting the trade-off between convergence speed and robustness in on- versus off-policy learning.

References

[1] Towers, M., Kwiatkowski, A., Terry, J., Balis, J. U., De Cola, G., Deleu, T., Goulão, M., Kallinteris, A., Krimmel, M., KG, A., et al., "Gymnasium: A Standard Interface for Reinforcement Learning Environments," *arXiv preprint arXiv:2407.17032*, 2024.

[2] Sutton, R. S., and Barto, A. G., *Reinforcement Learning: An Introduction*, A Bradford Book, Cambridge, MA, USA, 2018.

[3] Albrecht, S. V., Christianos, F., and Schäfer, L., *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*, MIT Press, 2024. URL `https://www.marl-book.com`.