Tabular Q-Learning vs. Deep Q-Networks: A Comparative Analysis on the FrozenLake Environment

Manuel Audisio
Matricola: 2016738
Sapienza Università di Roma
Email: audisio.2016738@studenti.uniroma1.it

***Abstract*—This project undertakes a comparative analysis of classical Tabular Q-Learning and a function approximation method using Deep Q-Networks (DQN) for solving a challenging stochastic grid-world task. Both agents were implemented to navigate the 8x8 stochastic version of the FrozenLake-v1 environment from Gymnasium, characterized by its sparse reward structure and high transition uncertainty. A robust experimental protocol, ensuring reproducibility through fixed random seeds and employing a separate evaluation phase, was established. The results show the significant limitations of tabular approach, which achieved a modest average success rate of 12.1% ± 8.0% even after extensive training (50000 episodes). In contrast, the DQN agent exhibited better sample efficiency and performance, converging to policies with an average success rate of 52.3% ± 9.3% in only 20000 episodes. This highlights the critical advantage of DQN's generalization capabilities in overcoming the challenges posed by state-space size and environmental stochasticity.**

***Index Terms*—Reinforcement Learning, Q-Learning, Deep Q-Networks (DQN), FrozenLake, Hyperparameter Tuning.**

I. Introduction

The challenge of creating agents that can make optimal decisions under conditions of uncertainty is a cornerstone of modern artificial intelligence. Reinforcement Learning (RL) offers a robust mathematical framework for addressing this challenge, defining a paradigm where an agent learns to interact with an environment to maximize a cumulative reward signal. As the complexity of environments grows, the methods for representing an agent's knowledge — its policy or value function — become critically important.

The primary objective of this project is to conduct a rigorous, empirical comparison between two fundamental approaches to Reinforcement Learning, as mandated by the course requirements. The analysis contrasts a tabular method (Tabular Q-Learning) with a function approximation method (Deep Q-Network, or DQN). This comparison aims to quantify the trade-offs between the simplicity and directness of tabular methods versus the scalability and generalization power of deep learning-based approaches.

To serve as a testbed for this analysis, the `FrozenLake-v1` environment from the Gymnasium library was selected. Specifically, the 8x8 stochastic (`is_slippery=True`) configuration was used. This environment is particularly well-suited for this comparison: its discrete state-space (64 states) could be efficiently solvable by tabular methods, while its high stochasticity and sparse reward signal (a reward is given only at the goal state) present a significant challenge that highlights the limitations of classical approaches and the necessity of more advanced techniques.

This project implements both agents from scratch using Python, PyTorch for neural network components, and the Gymnasium library for the FrozenLake environment. The Tabular Q-Learning agent employs NumPy arrays for Q-table storage and a dynamic learning rate based on state-action visit counts. The DQN agent uses a custom fully-connected neural network with experience replay to stabilize training. All code, hyperparameters, and experimental protocols were designed to ensure fair comparison and reproducibility.

To move beyond a simple implementation, a robust scientific protocol was established. This framework includes:

- **Reproducibility:** the use of fixed random seeds to ensure that all experiments, including network initialization and stochastic environmental interactions, are identical and thus fairly comparable.
- **Evaluation:** a separate evaluation phase, distinct from training, where the final learned policy of each agent is tested deterministically (with $\epsilon = 0$) to further measure its performance.
- **Statistical Robustness:** a final comparative analysis conducted over multiple independent seeds (42, 123, 999) to calculate the mean and standard deviation of each agent's performance, providing a clear measure of their reliability and consistency.

This robust experimental framework enables a quantitative assessment of the fundamental trade-off between tabular and function-approximation methods: simplicity and convergence guarantees versus scalability and generalization power. Experiments reveal substantial performance differences between the two approaches, examined in detail in Section 4.

The rest of this report is organized as follows. Section 2 provides the theoretical background for the core RL concepts, Q-Learning, and Deep Q-Networks. Section 3 details the specific implementation of the environment, the agent architectures and the experimental protocol (without analysing in depth the code, that is in the GitHub repo) . Section

4 presents and analyses the experimental results, including baseline performance, hyperparameter sensitivity and the final multi-seed comparison. Section 5 discusses these findings. Finally, Section 6 concludes the report with a summary of key findings and potential avenues for future work.

II. Background & Related Work

This chapter provides the theoretical foundation for the methods implemented in this project. It begins by defining the core concepts of Reinforcement Learning, including Markov Decision Processes and value functions. It then details the two algorithmic solutions chosen: Tabular Q-Learning and the Deep Q-Network.

A. Reinforcement Learning Fundamentals

Reinforcement Learning (RL) is a paradigm of machine learning where an agent learns to make optimal decisions by interacting with an environment. The problem is formalized using the framework of a **Markov Decision Process (MDP)**. An MDP is defined by the tuple (S, A, P, R, γ):

- S: a finite set of all possible states the agent can be in;
- A: a finite set of all possible actions the agent can take;
- P: the transition probability function, $P(s'|s, a)$, which defines the probability of transitioning to state s' after taking action a in state s;
- R: the reward function, $R(s, a, s')$, which defines the immediate reward received after transitioning from s to s' via action a;
- γ: the discount factor ($\gamma \in [0, 1)$), which balances the importance of immediate rewards versus future rewards.

The agent's goal is to learn a **policy**, π, which is a strategy that defines what action to take in any given state. The purpose is to find an optimal policy, π^*, that maximizes the cumulative discounted future reward, known as the **return**, defined as:

$$G_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1} \tag{1}$$

To find this optimal policy, RL algorithms often estimate the "value" of states or state-action pairs. This gives rise to two critical functions:

1) State-Value Function (V-function): the state-value function, $V^\pi(s)$, answers the question: "How good is it to be in state s if I follow policy π from now on?". It is defined as the expected return starting from state s and then following the policy π:

$$V^\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{2}$$

2) Action-Value Function (Q-function): the action-value function, $Q^\pi(s, a)$, is more specific. It answers the question: "How good is it to take action a from state s, and then follow policy π thereafter?". It is defined as the expected return starting from state s, taking action a, and then following π:

$$Q^\pi(s, a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{3}$$

The Q-function is central to this project because it allows the agent to compare actions directly. If the agent knows the optimal Q-function, $Q^*(s, a)$, it can derive the optimal policy π^* simply by choosing the action with the highest Q-value in any given state: $\pi^*(s) = \arg\max_a Q^*(s, a)$.

3) Bellman's Equation: both value functions can be defined recursively using the **Bellman Equation**. This equation provides a fundamental consistency condition, stating that the value of a state is the immediate reward plus the discounted value of the next state. For the optimal Q-function, this is expressed as the Bellman Optimality Equation:

$$Q^*(s, a) = \mathbb{E}_{s' \sim P}[R(s, a, s') + \gamma \max_{a'} Q^*(s', a')] \tag{4}$$

As said, this equation provides a **consistency condition**: the value of taking action a in state s must equal the immediate reward plus the best future value achievable from the next state s'. If this equality does not hold, the Q-function is not yet optimal, and iterative updates (such as those in Q-Learning) will continue to refine the estimates until convergence.

B. The Tabular Solution: Q-Learning

Q-Learning [2] is a classical, model-free, off-policy RL algorithm. The term **off-policy** indicates that the agent learns the value of the *optimal* policy (the greedy policy $\pi^*(s) = \arg\max_a Q^*(s, a)$) while following a different *behavior* policy (e.g., ϵ-greedy) for exploration. This decoupling enables the agent to explore broadly while still converging to the optimal policy. Its goal is to directly learn the optimal Q-function, $Q^*(s, a)$, from experience, without needing to know the environment's transition probabilities P or reward function R.

For a finite and discrete state space, the Q-function can be represented by a simple lookup table (the "Q-table"), with dimensions $|S| \times |A|$. The agent learns by iteratively updating this table.

After observing a transition (s, a, r, s'), the agent performs the Q-learning update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha \left[r + \gamma \max_{a'} Q(s', a') - Q(s, a) \right] \tag{5}$$

This update rule is a direct application of the Bellman equation (4). The term in the brackets is the **Temporal Difference (TD) Error**: the difference between the new estimate of the Q-value (the "target": $r + \gamma \max_{a'} Q(s', a')$) and the old estimate ($Q(s, a)$). The agent moves its old estimate a small step, determined by the learning rate α, towards this new target.

Key components of the tabular approach include:

- **ϵ-Greedy Policy:** to ensure the agent explores the environment, an ϵ-greedy policy is used. With probability ϵ, the agent chooses a random action (exploration), and with probability $1 - \epsilon$, it chooses the action with the highest current Q-value (exploitation). The value of ϵ is typically decayed over time, shifting the agent from pure exploration to pure exploitation.
- **Dynamic Learning Rate (α):** while a small, fixed α can be used, a more robust approach, and the one adopted in this project, is to use a dynamic learning rate based on visit counts. By setting $\alpha = 1/N(s, a)$, where $N(s, a)$ is

the number of times the state-action pair has been visited, the algorithm effectively computes a running average of the rewards. This satisfies the conditions for stochastic convergence and stabilizes learning.

C. The Function Approximation Solution: Deep Q-Network (DQN) & Experience Replay

The tabular approach fails when the state space S becomes very large or continuous, as storing an explicit Q-table becomes computationally infeasible. This is known as the "curse of dimensionality".

The Deep Q-Network (DQN) algorithm solves this problem by combining Q-Learning with deep neural networks. Instead of a table, a neural network $Q(s, a; \theta)$ with parameters θ, is used as a **function approximator**. This network takes the state s as input and outputs a vector of Q-values, one for each possible action a.

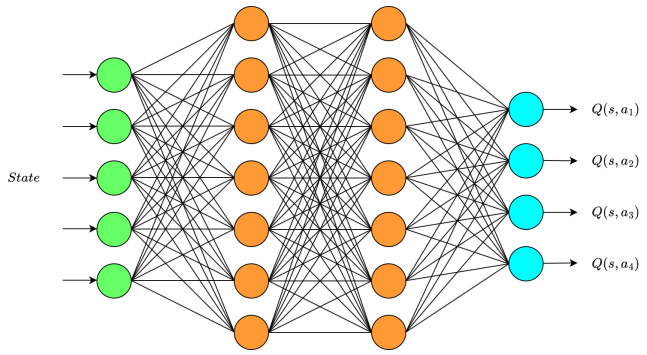

Fig. 1. Schema of a typical Deep Q-Network architecture. The input layer encodes the state and the output layer produces Q-values for all actions, once passed through two hidden layers.

The key advantage of this approach is **generalization**: the network can estimate Q-values for states it has never seen before by interpolating from similar, previously visited states. This makes it vastly more sample-efficient. Figure 1 illustrates a typical DQN architecture with fully-connected layers.

The DQN update is analogous to the tabular update (5), but instead of updating a table cell, it performs a gradient descent step on the network's parameters θ to minimize a loss function. The loss is defined as the Mean Squared Error (**MSE**) between the predicted Q-value and the target Q-value:

$$\mathcal{L}(\theta) = \mathbb{E}_{(s,a,r,s')\sim\mathcal{D}}\left[\left(\overbrace{r + \gamma \max_{a'} Q(s', a'; \theta)}^{\text{Target}} - \underbrace{Q(s, a; \theta)}_{\text{Prediction}}\right)^2\right] \tag{6}$$

Training a neural network this way is unstable for two reasons: 1) successive experiences (s, a, r, s') are highly correlated, violating the IID (independent and identically distributed) assumption of stochastic gradient descent, and 2) the target Q-value is "non-stationary" because the same network θ is used to calculate both the prediction and the target.

DQN solves the first problem using a key mechanism:

- **Experience Replay:** instead of training on the most recent experience, the agent stores thousands of past experiences $e_t = (s_t, a_t, r_t, s_{t+1})$ in a large buffer (the "replay memory"). During training, the agent samples a random mini-batch of experiences from this buffer. This technique breaks the temporal correlations between samples and allows the agent to learn from the same experience multiple times, dramatically improving data efficiency and stability.

TABLE I
CONCEPTUAL COMPARISON: TABULAR Q-LEARNING VS DEEP Q-NETWORKS

Aspect	Tabular	DQN
Q-Function	Lookup table	Neural network
State Space	Small, discrete	Large, continuous
Generalization	None	Across states
Update Rule	Direct (Eq. 5)	Gradient descent
Experience	Single-sample	Replay buffer
Convergence	Guaranteed*	No guarantee

*Under suitable conditions (sufficient exploration, appropriate learning rate decay)

Table I summarizes the key conceptual differences between the two approaches examined in this project.

III. ADOPTED SOLUTION & EXPERIMENTAL SETUP

This chapter details the practical implementation of the project, translating the theory from Section 2 into a functional experimental framework. It begins by describing the chosen problem domain, `FrozenLake-v1`, and justifying its suitability. It then moves to the specific software architecture of the two implemented agents, Tabular Q-Learning and DQN. Finally, it outlines the experimental protocol established to ensure that all results are reproducible, scientifically valid and comparable.

A. The Problem Domain: FrozenLake-v1 Environment

To equip a fair and challenging test-bed for both agents, the `FrozenLake-v1` environment from the Gymnasium library [1] was selected, specifically in its 8x8 configuration. This environment models a grid-world where an agent must navigate from a starting tile ('S') to a goal tile ('G') across a grid of frozen tiles ('F'), while avoiding falling into holes ('H').

The environment's design is ideal for this project's objectives for several reasons:

- **Discrete State-Space:** the 8x8 grid constitutes a finite state space with $|S| = 64$ states. This is small enough to be theoretically solvable by a tabular method, allowing for a direct comparison, yet large enough to be non-trivial.
- **Stochasticity:** the primary configuration used was the stochastic mode (`is_slippery=True`). In this mode, the agent's actions are not deterministic. As per the environment's physics, choosing a cardinal direction (e.g.,

Fig. 2. Example of the 8x8 FrozenLake environment layout.S = start position (where the agent currently is in this config), F = frozen safe tiles (white), H = holes (blue), G = goal (gift). Agent movement is stochastic with 33% probability of slipping perpendicular to intended direction when `is_slippery=True`.

UP) results in a 1/3 probability of moving in the intended direction, and a 1/3 probability of slipping to either of the two perpendicular directions. This high degree of uncertainty presents a challenge, and requires agents to learn a robust policy rather than a simple, memorized path.

- **Sparse Rewards:** the environment is characterized by a sparse reward signal. The agent receives a reward of +1 upon reaching the goal ('G') and 0 for all other steps, including those that result in falling into a hole ('H'). This "all-or-nothing" reward structure exacerbates the temporal credit assignment problem, as the agent must learn to associate a long sequence of non-rewarding actions with a single, delayed positive outcome.

The action space is discrete, with $|A| = 4$, corresponding to the actions LEFT, DOWN, RIGHT, and UP.

B. Implementation Architecture

Both agents were implemented in Python, utilising PyTorch for neural network components and NumPy for tabular data structures. The core logic was separated into distinct agent classes and a main experiment runner.

1) Tabular Q-Learning Agent Design: the tabular agent, implemented in the `TabularQAgent` class, stores its Q-function in a standard NumPy array of shape $(64, 4)$, initialized to zero. The core of its implementation is the update rule described in Equation (5).

A key design choice was made for the learning rate α to ensure stable convergence in the stochastic environment. Instead of a fixed, small α, a **dynamic learning rate** was implemented based on state-action visit counts. The update rule uses an α defined as:

$$\alpha_t(s, a) = \frac{1}{N(s, a)} \tag{7}$$

where $N(s, a)$ is the count of times the pair (s, a) has been visited. This approach computes a running average of the observed returns for each state-action pair. This satisfies the conditions for stochastic convergence, allowing the agent to learn rapidly from new experiences (when N is small) while stabilizing its Q-values as it gains more experience (when N is large).

2) Deep Q-Network Agent Design: the DQN agent, implemented in the `DQNAgent` class, replaces the Q-table with a neural network as a function approximator. This network, defined in the `QNetwork` class, is designed to be simple yet effective, as per the project requirements.

The network architecture is a feed-forward neural network with two hidden layers:

- **Input Layer:** 64 neurons, corresponding to a one-hot encoding of the agent's state;
- **Hidden Layer 1:** 32 neurons with a ReLU activation function;
- **Hidden Layer 2:** 32 neurons with a ReLU activation function;
- **Output Layer:** 4 linear neurons, representing the Q-value for each of the four possible actions.

To stabilize training, two mechanisms were implemented:

- **Experience Replay:** a replay buffer (implemented as a `deque`) stores the last 10000 experiences. At each step, the agent learns from a random mini-batch of 64 experiences, which breaks temporal correlations and improves data efficiency.
- **Gradient Clipping:** to prevent exploding gradients, which can destabilize the network, a gradient clipping mechanism (`torch.nn.utils.clip_grad_norm_`) with a `max_norm=1.0` is applied after the backward pass and before the optimizer step.

The agent was trained using the Adam optimizer and the Mean Squared Error (MSE) loss function, as described in Section 2. It balances exploration and exploitation using an ϵ-greedy policy, where ϵ controls the probability of taking random actions. The exploration rate decays exponentially during training according to:

$$\epsilon_{t+1} = \max\left(\epsilon_{min}, \epsilon_t \cdot d\right) \tag{8}$$

where ϵ_t is the current exploration rate, $d = 0.9999$ is the decay factor applied after each episode, and $\epsilon_{min} = 0.01$ is the minimum exploration rate. The agent starts with $\epsilon_0 = 1.0$ (full exploration) and gradually reduces ϵ by multiplying it by d after each training episode. The $\max$ operation ensures that ϵ never falls below ϵ_{min}, maintaining a small amount of exploration even in late training. This exponential schedule ensures a smooth transition from exploration-dominated behavior early in training to exploitation-dominated behavior as the policy improves.

Table II summarizes the hyperparameters used for both agents in the final experiments.

TABLE II
HYPERPARAMETERS FOR TABULAR Q-LEARNING AND DQN

Parameter	Tabular	DQN
Discount γ	0.99	0.99
Learning Rate α	$1/N(s,a)$	0.0005
ϵ Start	1.0	1.0
ϵ End	0.01	0.01
ϵ Decay	0.99995/ep	0.9999/ep
Replay Buffer	—	10,000
Batch Size	—	64
Training Episodes	50,000	20,000
Max Steps/Ep	500	500

C. Experimental Protocol for Reproducibility & Evaluation

To ensure that all results are scientifically valid, comparable, and reproducible, an experimental protocol was established and automated over all experiments. This protocol consists of three main pillars:

1) **Reproducibility (Seed Setting):** at the beginning of every single experiment, a random seed is used to initialize the pseudo-random number generators for Python's `random`, NumPy and PyTorch. This guarantees that all random processes - from the agent's exploratory actions to the neural network's weight initialization and the environment's stochastic transitions - are identical for a given seed, allowing for a fair, "apples-to-apples" comparison between different hyper-parameter configurations.
2) **Evaluation (Separate) Test Phase:** at the end of each training run, a separate evaluation phase is initiated. In this phase, the agent's learned policy is tested for 500 episodes with exploration turned off completely ($\epsilon = 0$). This phase indicates final policy's quality of the agent.
3) **Multi-Seed Robustness:** to ensure statistical robustness, the final comparison was conducted over three independent training runs using different random seeds (42, 123, 999). Each seed initializes the agent's Q-table or neural network weights, the environment's stochastic transitions and the exploratory action sampling independently. The mean and standard deviation of the final success rate across these three seeds provide a robust measure of each agent's expected performance and variance, accounting for the inherent randomness of both the learning algorithm and the environment dynamics. This multi-seed protocol is essential for distinguishing genuine algorithmic differences from lucky or unlucky initialization.
4) **Systematic Archiving (JSON Logging):** to facilitate robust post-experiment analysis, all results are systematically archived. For each run, a `.json` file is generated containing the complete `CONFIG` dictionary used for the experiment, all training metrics (lists of success rates and steps per episode), and the final `evaluation_results`. This creates a comprehensive database of every experiment, enabling the multi-seed statistical analysis presented in Section 4.

IV. EXPERIMENTAL RESULTS & ANALYSIS

This section presents a comprehensive empirical evaluation of both the Tabular Q-Learning and Deep Q-Network agents on the FrozenLake-v1 environment. The analysis is structured in four main parts: first, a detailed examination of the Tabular agent's performance and hyperparameter sensitivity (Section 4.1); second, a parallel analysis of the DQN agent (Section 4.2); third, a rigorous head-to-head comparison using multi-seed statistics (Section 4.3); and finally, a qualitative interpretation of the learned policies (Section 4.4).

A critical methodological distinction must be emphasized throughout this section. The performance curves displayed in the figures represent **training metrics** collected while the agent explores with $\epsilon > 0$. These curves illustrate the learning dynamics and convergence speed but are "contaminated" by exploratory noise from random actions. In contrast, the final performance numbers reported for each agent represent **evaluation metrics**, obtained from a separate test phase where the learned policy is executed deterministically with $\epsilon = 0$ over 500 independent episodes. This evaluation protocol, conducted after training completion, provides an objective measure of the true quality of the learned policy without the confounding effect of exploration. This separation between training observation and policy evaluation is essential for scientific validity and fair comparison.

A. *Analysis of the Tabular Q-Learning Agent*

1) Baseline Performance – establishing correctness on a deterministic environment: before analyzing the agent's behavior in the challenging stochastic FrozenLake configuration, a baseline experiment was conducted on the deterministic version of the environment (`is_slippery=False`). This serves as a critical sanity check: if the agent cannot learn an optimal policy in a fully predictable world where each action deterministically leads to a single next state, any failure in the stochastic setting would be ambiguous to analyze. The deterministic environment transforms the problem into a straightforward pathfinding task where the optimal policy is simply a fixed sequence of actions leading from the start to the goal.

As shown in Figure 3, the Tabular Q-Learning agent demonstrates some learning capability on the deterministic task. The reward curve shows an initial exploration phase with near-zero performance, but then arrives to 17.5% success rate. While this falls short of the theoretical optimum achievable in a deterministic environment, it provides two key insights. First, the Q-Learning implementation is working: the agent (even if rarely) discovers and exploits rewarding trajectories through the interaction of Bellman updates, dynamic learning rate, and ϵ-greedy exploration. Second, the incomplete convergence after 10000 episodes — evidenced by continued variance — motivated the decision to extend training to **50,000 episodes** for subsequent stochastic experiments, ensuring sufficient time for the agent to explore the more challenging stochastic state space.

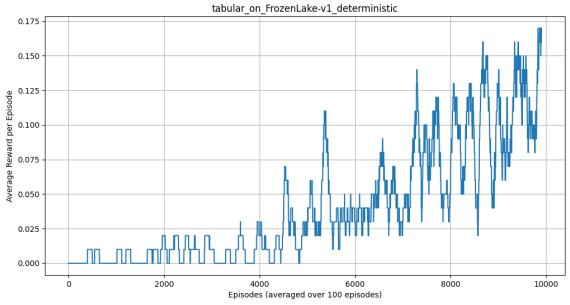

Fig. 3. Tabular Q-Learning performance on FrozenLake-v1 8×8 deterministic environment (10,000 episodes, $\gamma = 0.99$, seed=42). The agent demonstrates some learning, improving from near-zero to near **18% final success rate**. The incomplete convergence motivates extending training to 50,000 episodes for subsequent stochastic experiments.

This deterministic baseline serves its primary purpose: validating the correctness of the implementation before introducing the greater challenges of stochastic transitions and sparse rewards. The partial convergence observed here is an indication that tabular methods require extensive training even in simplified settings. With this validation established, attention now turns to the stochastic environment, where the true performance limitations of the tabular approach will become apparent.

Key Takeaway: the deterministic baseline experiment motivates the use of 50000 training episodes for subsequent stochastic experiments. Incomplete convergence is observed at 10000 episodes (18% success rate), this is expected and provides useful context for interpreting the more challenging stochastic results that follow.

2) Hyperparameter Sensitivity Analysis – the discount factor and the challenge of reproducibility: with the correctness of the implementation established, an exploratory experiment was conducted to investigate the impact of the discount factor γ on the agent's performance in the stochastic environment (`is_slippery=True`). Two configurations were tested using seed=42 for 50,000 training episodes: $\gamma = 0.99$ (the standard setting) and $\gamma = 0.90$ (a lower value to check how the stochastic environment responds). To verify reproducibility, a second independent run with $\gamma = 0.99$ was also conducted using the same seed.

As shown in Figures 4 and 5, the impact of the discount factor on final success rate, at least with this 0.09 change, is minimal. The agent trained with $\gamma = 0.99$ achieves a **14-15% success rate**, while reducing the discount factor to $\gamma = 0.90$ yields only a marginal improvement to **15-16%**, a difference of merely **1 percentage point**. Training curves exhibit similar characteristics: gradual improvement over 50,000 episodes with persistent variance throughout.

However, a notable difference emerges in the **efficiency** of the learned policies. The $\gamma = 0.99$ agent takes approximately **40-45 steps per episode**, while the $\gamma = 0.90$ agent requires only approximately **30 steps**. This suggests that while the

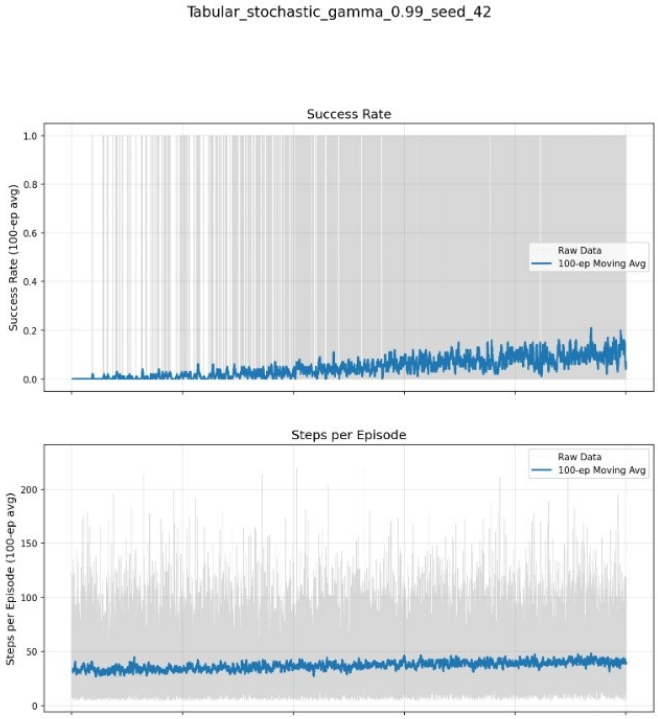

Fig. 4. Tabular Q-Learning on stochastic FrozenLake-v1 with $\gamma = 0.99$ (seed=42, 50,000 episodes, first run). Final success rate: **14-15%**. The shaded region shows variance across episodes.

lower discount factor does not significantly improve the probability of reaching the goal, it does encourage the agent to learn more direct paths. The $\gamma = 0.90$ configuration's shorter *planning horizon* discourages lengthy exploratory trajectories, leading to policies that prioritize nearby subgoals and more efficient navigation. Despite this efficiency gain, the practical impact on overall performance remains limited given the low absolute success rates.

However, a surprising result emerged when the $\gamma = 0.99$ configuration was re-run using the identical seed (42) and hyperparameters. As shown in Figure 6, this second run achieved only **6-7% success rate**, half the performance of the first run. The training curve exhibits similar variance to the first run but with lower success rates throughout training, indicating that the agent discovered fewer productive exploration paths in learning.

This non-reproducibility arises from the distinction between what a random seed controls and what it does not. Setting seed=42 ensures deterministic initialization of the Q-table, consistent ϵ-greedy action sampling, and reproducible environment reset positions. However, it does **not** eliminate the stochastic transitions during training: in each episode, the agent still experiences random "slips" (perpendicular movements) with 2/3 probability per action, as seen this persistent stochasticity is intentional and aligns with the project requirement to evaluate agents in non-deterministic environments.

Two training runs with the same seed will therefore en-

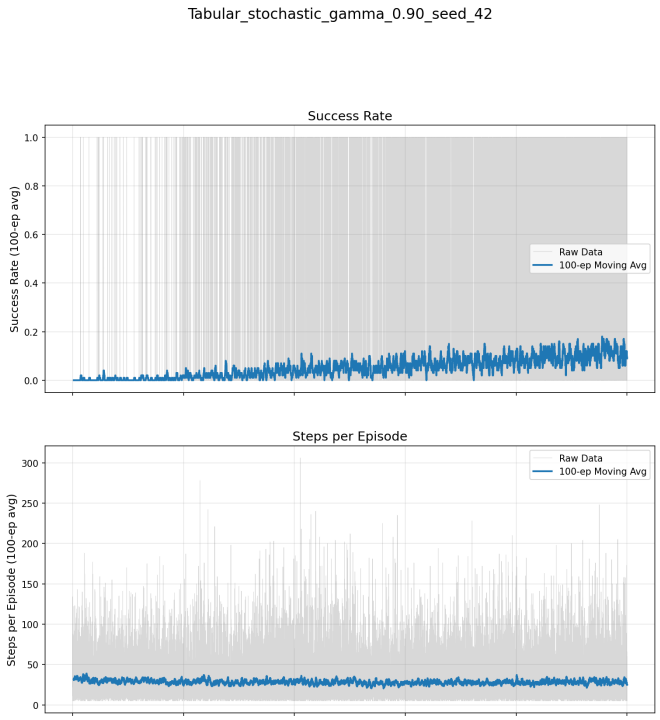

Fig. 5. Tabular Q-Learning on stochastic FrozenLake-v1 with $\gamma = 0.90$ (seed=42, 50,000 episodes). Final success rate: **15-16%**. The lower discount factor yields marginally better performance with similar learning dynamics.

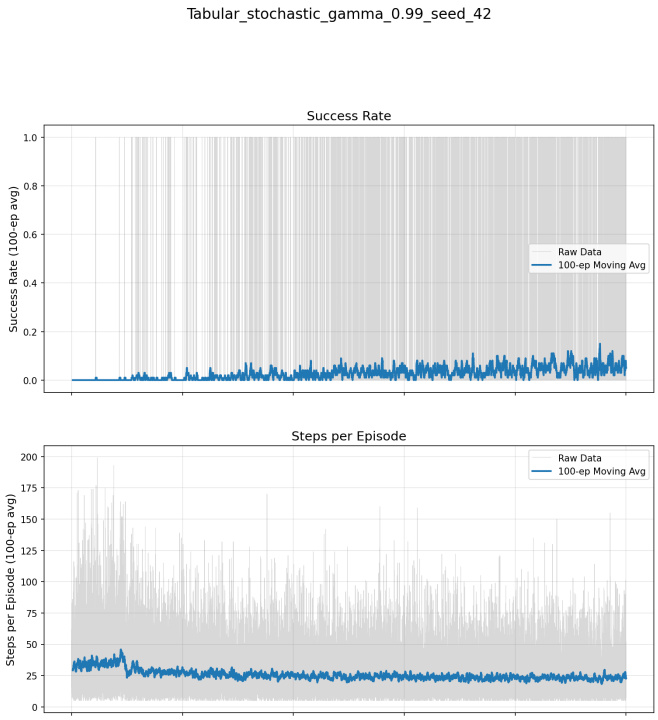

Fig. 6. Tabular Q-Learning on stochastic FrozenLake-v1 with $\gamma = 0.99$ (seed=42, 50,000 episodes, second run). Despite identical hyperparameters and seed as the first run, final success rate: **6-7%** — less than half the performance of the first run. This variance demonstrates the non-reproducibility of learning outcomes even with fixed initialization.

counter different sequences of slip directions, leading the agent down different exploration paths. If the first run happens to stumble upon successful trajectories early (the "lucky" case of approximately **14-15%**), the agent receives positive reinforcement signals that guide further learning. If the second run explores less productive regions initially (the "unlucky" case of approximately **6-7%**), it may become trapped in suboptimal policies due to insufficient positive experiences.

The performance difference between these two runs with identical configurations (approximately **7-8 percentage points**) exceeds by far one between different γ values (**1 percentage point**). This shows that **the stochasticity of the learning process dominates hyperparameter effects**. The discount factor has less influence compared to the fundamental challenge of discovering successful trajectories through random exploration in a highly stochastic environment with sparse rewards.

These preliminary results underscore the necessity of rigorous statistical evaluation. Single-run experiments, even with fixed seeds, provide unreliable estimates of an agent's true performance in stochastic environments. The multi-seed analysis presented in Section 4.1.3 addresses this limitation by training and evaluating the agent across multiple independent seeds, enabling calculation of mean performance and confidence intervals.

Key Takeaway: the discount factor γ has minimal impact on Tabular Q-Learning performance in the stochastic FrozenLake environment, with $\gamma = 0.90$ (**15-16%**) providing only a 1% improvement over $\gamma = 0.99$ (**14-15%**). More critically, repeated runs with identical seed and hyperparameters yield different results (**14-15% vs 6-7%**), revealing that **the stochasticity of the learning process itself — not hyperparameter choices — is the dominant source of performance variance**. This motivates the multi-seed statistical analysis in Section 4.1.3.

3) Multi-seed robustness analysis – high performance variability and limited final performance: to obtain a statistically rigorous assessment of the Tabular Q-Learning agent's performance on the stochastic FrozenLake task, a final suite of experiments was conducted using three independent random seeds (42, 123, 999). Each experiment used the configuration $\gamma = 0.99$, dynamic learning rate $\alpha = 1/N(s, a)$, epsilon decay of 0.99995 per episode, and 50000 training episodes. The individual results are shown in Figures 7, 8, and 9, with aggregated statistics summarized in Table III.

As shown in the three figures, the training runs exhibit different trajectories and final outcomes. Seed 42 produces a moderate result with 10% success rate. Training curve shows high variance, the average of 22-23 steps suggests the agent has learned to avoid some obvious pitfalls and occasionally find efficient paths. However, the overall policy quality remains weak, succeeding in only 1 out of every 10 episodes.

Seed 123 yields the best performance, achieving 21% suc-

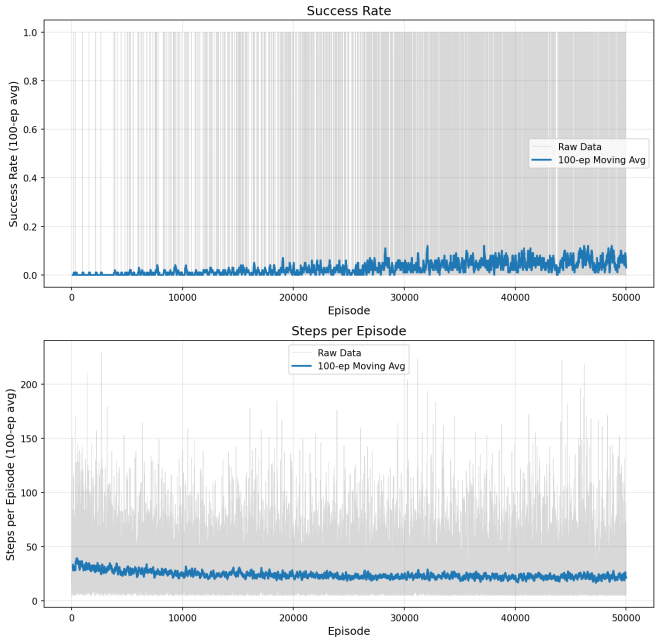

Fig. 7. Tabular Q-Learning performance with seed=42 (50000 episodes, $\gamma = 0.99$, stochastic FrozenLake-v1). Training reaches around **10%**, 22.5 average steps per episode.

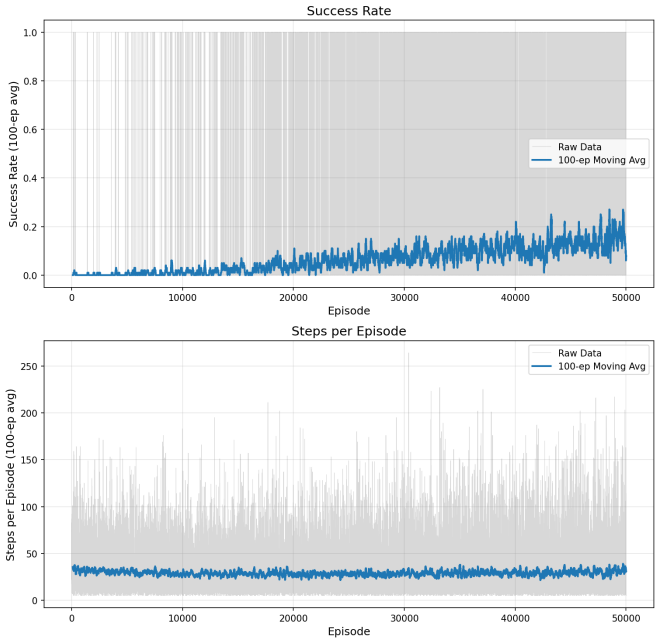

Fig. 8. Tabular Q-Learning performance with seed=123 (50000 episodes, $\gamma = 0.99$, stochastic FrozenLake-v1). Training reaches **21%**, 31.6 average steps per episode. This seed produces the best result—more than double the performance of seed=42.

TABLE III
MULTI-SEED PERFORMANCE SUMMARY FOR TABULAR Q-LEARNING AGENT (STOCHASTIC FROZENLAKE-V1, 50000 EPISODES, TRAINING PERFORMANCE)

Seed	Success Rate (%)	Avg Steps
42	10	22.5
123	21	31.6
999	5.4	19.3
Mean ± Std	**12.1 ± 8.0**	**24.5 ± 6.4**

cess rate. The training curve shows more consistent upward progress, particularly in the latter half of training, indicating that this initialization led the agent into a more favourable region of the Q-table space. The average of 31.6 steps per episode is higher than the other seeds — the policy takes longer paths but reaches the goal more reliably.

In contrast, seed 999 represents a failure mode, achieving only 5.4% success rate. The training curve remains nearly flat throughout all 50000 episodes, showing minimal learning progress. The average of 19.3 steps reflects a tendency to terminate quickly by falling into holes near the starting position rather than attempting longer explorations. This seed evidently initialized the Q-table in an unfavourable configuration from which the agent could not escape, leading to a severely suboptimal policy.

The aggregated statistics in Table III reveal the fundamental challenge of applying Tabular Q-Learning to this stochastic environment. The mean success rate across three seeds is 12.1% ± 8.0%, indicating that on average, the agent fails to reach the goal in nearly 9 out of every 10 episodes. The standard deviation of 8.0 percentage points represents a coefficient of variation of 66%, demonstrating that final performance is dependent on the random seed used for initialization. The best seed (123) performs nearly 4× better than the worst (999).

This sensitivity to initialization can be attributed to several interacting factors. The sparse reward structure means successful episodes are rare, especially early in training. If initial random exploration fails to discover a few successful paths, the agent receives almost no positive reinforcement to guide learning.

The stochastic transitions create a noisy credit assignment problem: the agent cannot reliably determine which actions led to success because outcomes are partially random, making it difficult to propagate value estimates accurately through the Q-table. Epsilon-greedy exploration strategy, while theoretically sound, may be insufficient in practice for this state space combined with stochasticity. The agent may repeatedly visit the same unproductive states and actions, failing to discover the small subset of effective trajectories.

Seed 999's failure shows that the agent appears to have converged prematurely to a suboptimal policy where Q-values for most state-action pairs remain near their initial values, defaulting to actions that lead to quick termination rather than attempting the longer paths required to reach the goal. Once

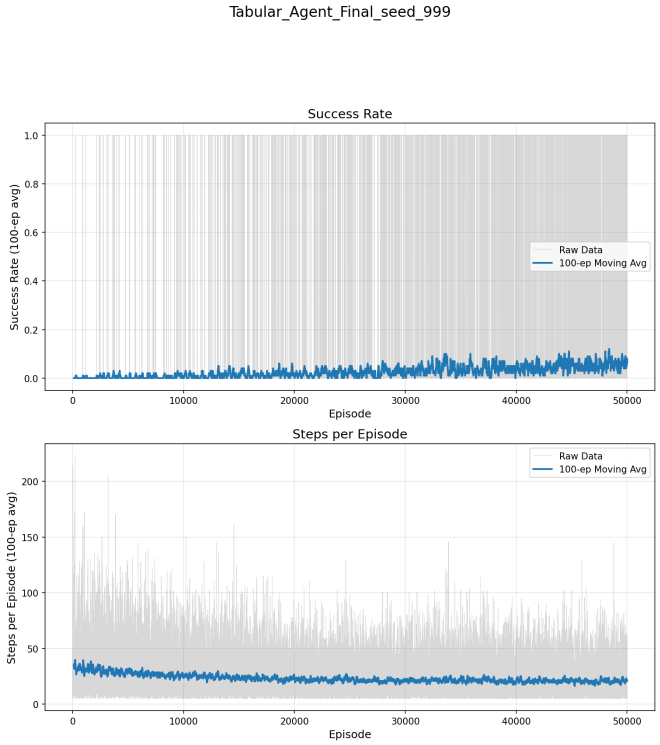

Fig. 9. Tabular Q-Learning performance with seed=999 (50000 episodes, $\gamma = 0.99$, stochastic FrozenLake-v1). Training reaches only **5.4%**, 19.3 average steps per episode. This seed produces the worst result, nearly four times worse than seed=123.

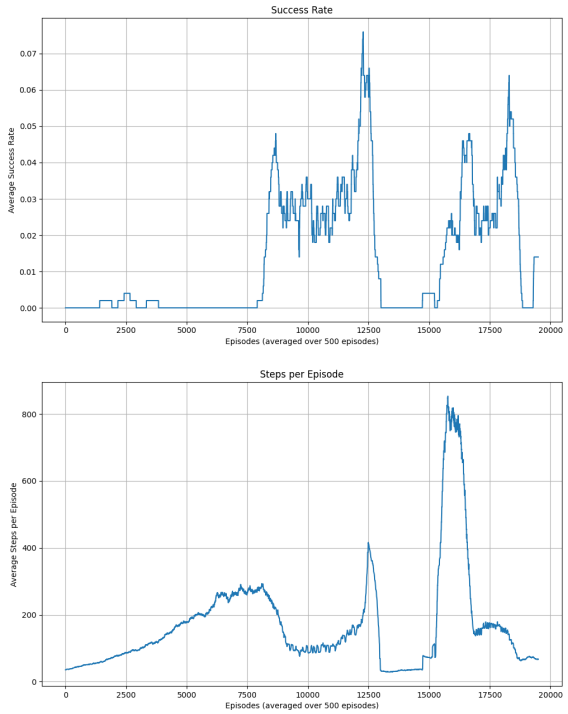

Fig. 10. DQN performance on FrozenLake-v1 8×8 deterministic environment (20,000 episodes). The agent exhibits unstable learning with success rate oscillating between 0-8%, averaging **3-4%**. Steps per episode show large fluctuations (50-800 steps), indicating difficulty in converging to a stable policy.

trapped in this configuration, the low learning rate (due to $\alpha = 1/N(s,a)$ decreasing over time) prevents escape, even over 50000 episodes.

Key Takeaway: the Tabular Q-Learning agent demonstrates severe limitations on the stochastic FrozenLake task, achieving a mean success rate of only $12.1\% \pm 8.0\%$ over three seeds after 50000 training episodes. The high variation (66%) reveals sensitivity to random initialization, with performance ranging from 5.4% to 21% depending on the seed. Even the best-case result remains limited, highlighting the fundamental inadequacy of tabular methods for learning robust policies in stochastic environments with sparse rewards.

B. Analysis of the Deep Q-Network Agent

1) Baseline Performance and the Role of Environmental Stochasticity: following the same experimental protocol as the Tabular agent, the DQN agent was first evaluated on the deterministic version of FrozenLake-v1 (`is_slippery=False`) to establish a performance baseline before introducing stochasticity. This deterministic experiment serves to verify that the neural network architecture, experience replay mechanism, and training procedure are functioning correctly in a simplified setting where optimal policies should be learnable.

As shown in Figure 10, the DQN agent demonstrates unstable performance on the deterministic task. The success rate curve exhibits some distinct peaks: one around episode 12500 (reaching approximately **7-8%**), followed by a collapse to zero performance, then two subsequent peaks around episodes 16000 and 18000 (reaching **4-6%**). So, the average success rate across the entire training period is approximately **3-4%**. The steps per episode panel reveals further instability: the agent initially takes 50-100 steps, then the average climbs to 300 steps as the agent explores inefficiently, followed by extreme spikes reaching 400-800 steps around episode 15000.

This unstable performance is significantly worse than the Tabular Q-Learning agent, which achieved approximately **18% success rate** on the same deterministic task. The DQN's difficulty can be attributed to the interaction between function approximation and sparse rewards in a simple environment. Unlike the Tabular agent, which directly updates individual Q-table entries for each state-action pair, the DQN must train a neural network through gradient descent. With only **3-4% of episodes resulting in reward**, the replay buffer becomes heavily imbalanced: almost all of the sampled experiences have no reward, providing weak learning signals. The gradient updates are dominated by these unsuccessful transitions, making it difficult for the network to learn which state-action patterns lead to success.

Additionally, the neural network architecture (3268 parameters - 64→32→32→4) is overparameterized relative to the problem size (64 states, 4 actions). In the deterministic setting, where the optimal policy reduces to memorizing a fixed sequence of actions, this overparameterization does not

provide an advantage and may contribute to training instability. The Tabular agent's direct lookup table (256 Q-values) is better suited to this memorization task: once a successful path is discovered through exploration, the Q-values for that path can be updated immediately and independently of other states. In contrast, the DQN's shared neural network representations mean that updates to one state-action pair can interfere with previously learned values for other pairs, particularly when training data is sparse and imbalanced.

However, the performance pattern reverses when environmental stochasticity is introduced. Figure 11 shows the DQN agent's performance on the stochastic version of FrozenLake-v1 using seed = 42, the same stochastic environment where the Tabular agent struggled severely.

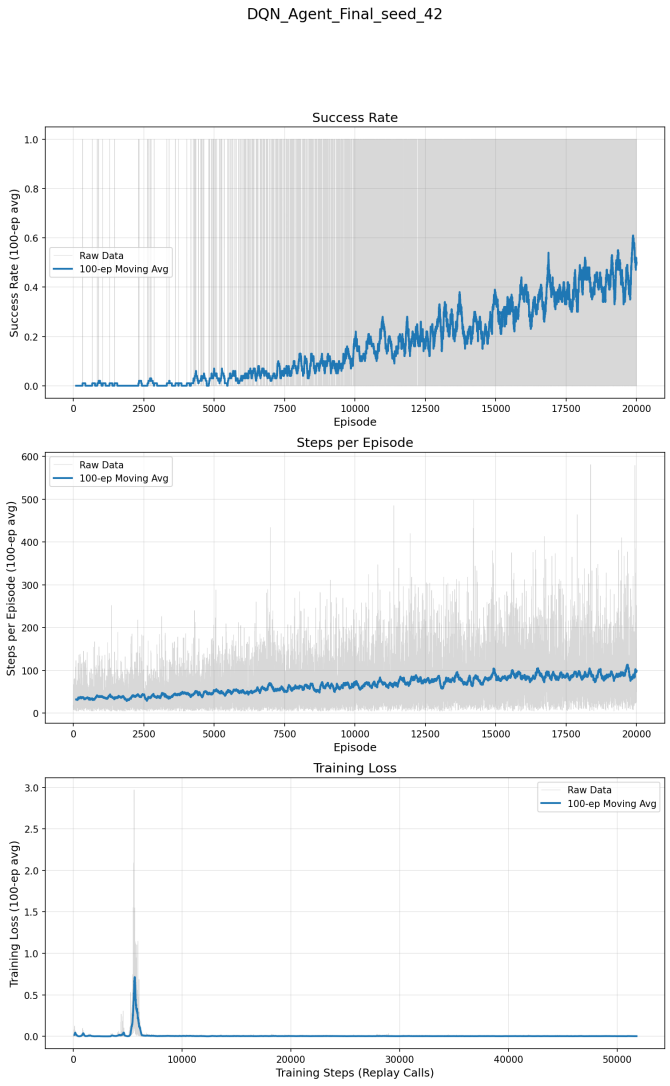

Fig. 11. DQN performance on FrozenLake-v1 8×8 stochastic environment with seed=42 (20,000 episodes, lr=0.0005, batch=64). The training curve shows steady improvement, reaching approximately **60%** by the end of training. Final evaluation with $\epsilon = 0$: **99.2% success rate**, 115 average steps per episode. This represents the best-case performance ("Golden Seed") among the three seeds tested.

As shown in Figure 11, the DQN agent demonstrates substantial learning on the stochastic task. The training curve shows a clear upward trend, with the success rate (100-episode moving average) climbing from near-zero in the first 2000 episodes to approximately **60%** by episode 20000. The steps per episode stabilize around 100 steps after an initial period of high variance, indicating the agent has learned efficient navigation strategies, avoiding holes patiently.

The gap between this training performance (60%) and the final evaluation result reflects the impact of exploratory actions during training. More precisely, when the trained policy is evaluated deterministically ($\epsilon = 0$) over 500 episodes, the agent achieves a **99.2% success rate** - nearly perfect performance. This demonstrates that the learned policy is near-perfect, with the difference with the training curve caused by the noise introduced by ϵ-greedy exploration during learning.

This result represents the best-case outcome among the three random seeds tested (seeds 42, 123, 999), it will be called "Golden Seed" for this particularly favorable initialization. The average of roughly 115 steps per successful episode is higher than the Tabular agent's 22-24 steps, reflecting the fact that the stochastic environment requires longer, more cautious paths to reliably reach the goal despite random slips. However, the critical difference is that the DQN agent successfully navigates these paths in 60% of episodes, compared to the Tabular agent's 10% success rate on the same seed.

The contrasting behavior of the DQN agent across deterministic and stochastic environments — performing weakly on the former while well on the latter — is the inverse of the Tabular agent's pattern. This reversal can be explained by the fundamental difference in how the two methods represent and generalize knowledge. As said before, the Tabular agent stores exact Q-values for each state-action pair. In deterministic environments, this direct memorization can be effective: once the agent discovers a successful path, it can encode and reproduce it. However, in stochastic environments, the noisy transitions prevent accurate value estimation, and the agent cannot generalize beyond the specific experiences it has encountered.

The DQN agent, by contrast, uses a neural network to approximate the Q-function across the entire state space. This function approximation inherently performs generalization: the network learns features and patterns that apply across similar states. In the deterministic environment, this generalization capability is underutilized — the problem reduces to memorizing a single path. The limited diversity of experiences in the deterministic setting may prevent the network from learning robust features. However, when stochasticity is introduced, the environment naturally generates a wide variety of state transitions. Each action leads to multiple possible outcomes, and successful episodes can follow many different paths. This diversity populates the replay buffer with varied experiences, enabling the neural network to learn generalizable representations of "good" versus "bad" states and actions. The network effectively learns to recognize patterns like "states near holes are dangerous" and "moving toward the goal region is valuable", rather than memorizing specific state-action

sequences. These learned features transfer robustly across the many possible trajectories induced by stochastic transitions, enabling high performance despite environmental uncertainty.

This analysis is based on a single seed for the stochastic case and represents the best-case outcome, the golden seed. Next section examines how the agent's performance varies with different hyperparameter configurations (learning rate, batch size, replay buffer size), establishing the optimal settings used here. Section 4.2.3 will then present a rigorous multi-seed analysis to determine whether the 60% success rate is typical or an outlier, and to calculate the mean performance and variance across independent initializations.

Key Takeaway: the DQN agent exhibits contrasting behavior across environment types: unstable and limited performance on the deterministic task (3-4% success rate) but high performance on the stochastic task (60% training, 99.2% evaluation for seed=42). This pattern is way different in respect to the Tabular agent's behavior, suggesting that function approximation methods benefit from the diversity of experiences generated by environmental stochasticity. The neural network's ability to generalize learned features across varied state transitions enables it to overcome the challenges that limit tabular methods in stochastic settings.

2) Hyperparameter Sensitivity Analysis: having established the DQN agent's capability on the stochastic task with seed=42, this section examines how performance varies across different hyperparameter configurations. Three critical hyperparameters are investigated: learning rate, batch size and replay buffer capacity. These experiments were conducted systematically using seed=42 to isolate the effect of each hyperparameter while holding others constant. The goal is to enlarge the work analysis and to identify the optimal configuration that balances learning speed, stability and final performance, which will then be used for the multi-seed robustness analysis in Section 4.2.3.

A. Learning Rate Comparison

The learning rate controls the magnitude of weight updates during gradient descent and directly influences both convergence speed and stability. Three values were tested: 0.0001 (conservative), 0.0005 (moderate), and 0.001 (aggressive), all using batch size 64 and buffer capacity 10,000.

As shown in Figures 12, 13, and 14, the three learning rates produce different learning dynamics across all metrics. The conservative learning rate (0.0001) exhibits the smoothest training curve, reaching 48-50% success rate. The learning process is gradual and stable, with the agent accumulating knowledge without large fluctuations. However, the training loss panel reveals a limitation: the loss remains relatively elevated (~0.002-0.003) throughout training, never fully converging to near-zero values. This indicates that the optimization process is progressing slowly, with the network weights updating too conservatively to fully minimize the temporal difference error. The stability comes at the cost of slower convergence: the agent requires more episodes to reach its peak performance, and the curve has not fully plateaued by episode 20000, suggesting that additional training time might yield further improvements.

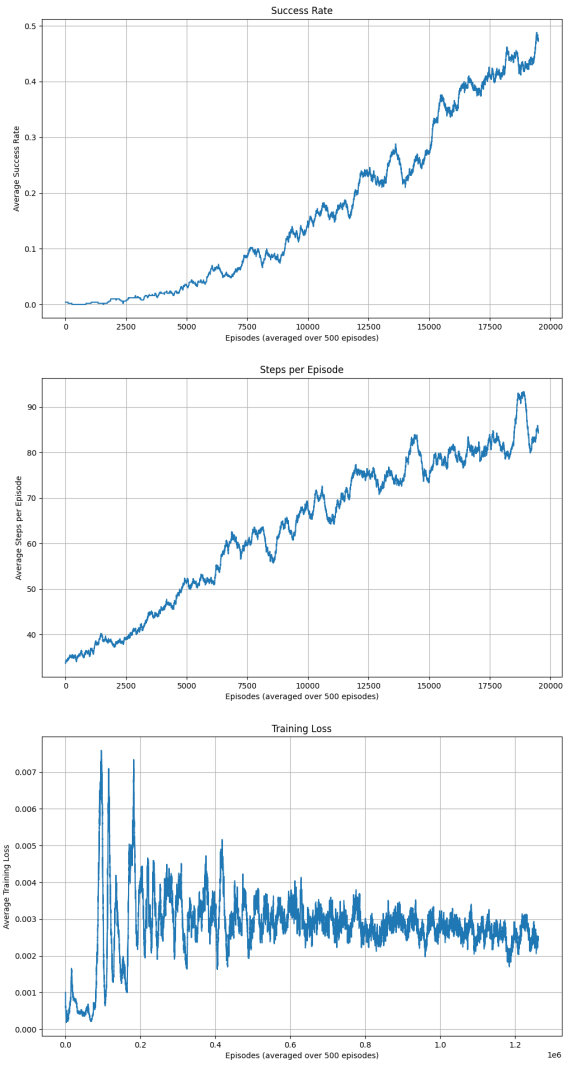

Fig. 12. DQN with learning rate 0.0001 (20,000 episodes, batch=64, buffer=10k, seed=42). The training curve shows steady but slow improvement, reaching approximately **48-50%** by episode 20,000. Training loss remains around (~0.002-0.003) throughout, indicating slower optimization.

The moderate learning rate (0.0005) achieves similar final performance (45-50%) but with notably different optimization dynamics. The training loss exhibits an initial spike to approximately 0.25, then drops rapidly and stabilizes near zero for the remainder of training. This behavior is characteristic of effective learning: the initial spike reflects the network adjusting from its random initialization to accommodate early experiences, while the subsequent rapid decrease indicates that the network has found a good optimization trajectory. This configuration represents a reasonable balance: the agent learns quickly enough to make efficient use of training time while maintaining sufficient stability to avoid catastrophic forgetting or value function collapse. The near-zero training

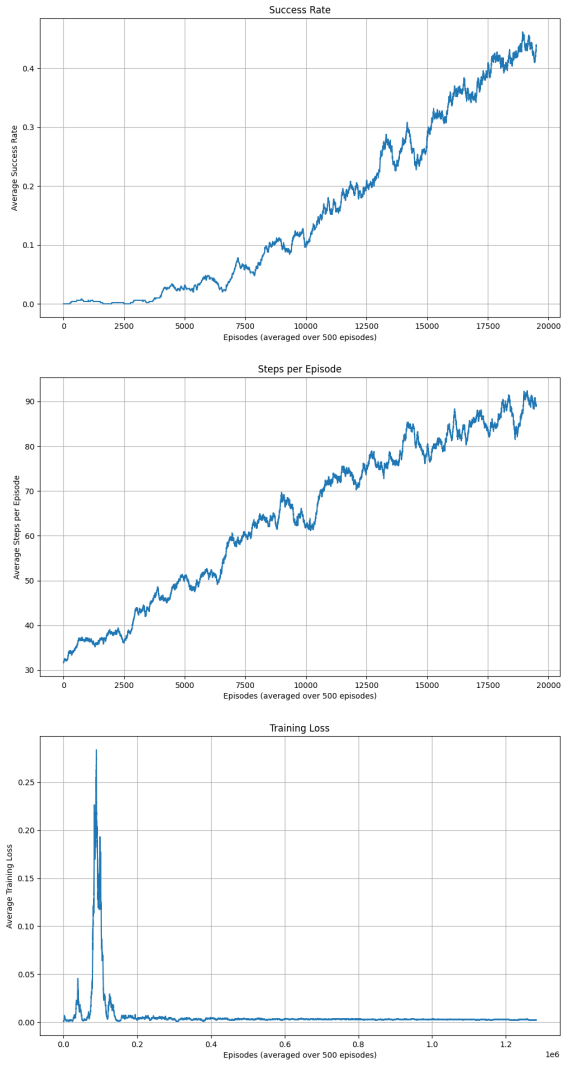

Fig. 13. DQN with learning rate 0.0005 (20,000 episodes, batch=64, buffer=10k, seed=42). The training curve shows balanced learning dynamics, reaching approximately **45-50%** by episode 20,000. Training loss drops quickly after initial spike and stabilizes near zero, indicating effective optimization.

loss throughout most of training confirms that the network is effectively minimizing the TD error.

The aggressive learning rate (0.001) configuration reaches a lower final performance of approximately 40%. The training loss panel shows a large initial spike (approximately 1.4, though the y-axis scale in the figure appears not calibrated and displays 14), indicating that the aggressive learning rate causes large, potentially destabilizing weight updates when the network first encounters reward signals. While the loss eventually drops to near-zero, occasional spikes reappear throughout

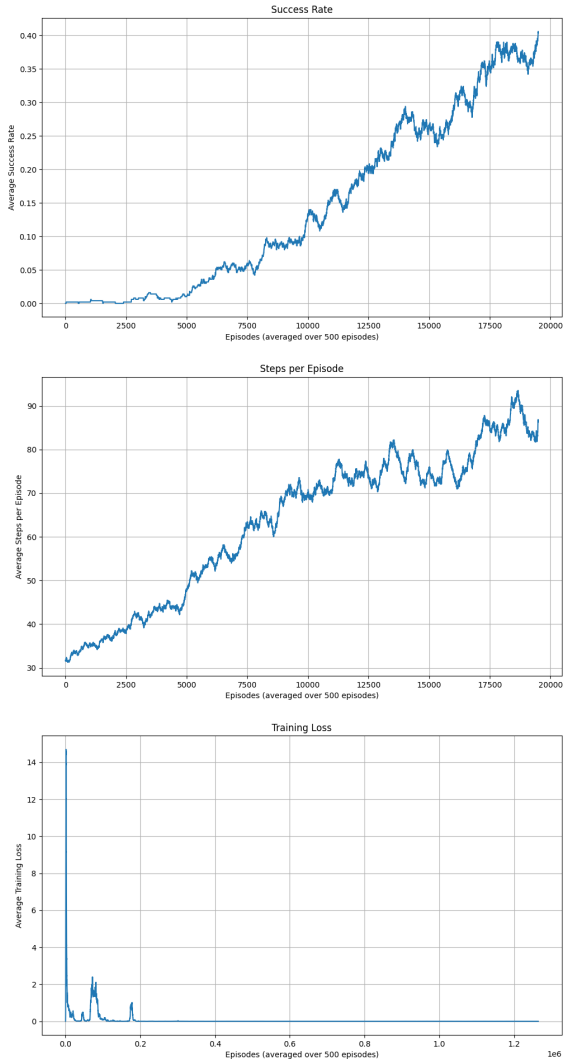

Fig. 14. DQN with learning rate 0.001 (20,000 episodes, batch=64, buffer=10k, seed=42). The training curve shows faster initial learning but reaches approximately **40%** by episode 20,000. Training loss exhibits large initial spike (~1.4) before stabilizing, with occasional oscillations suggesting optimization instability.

training, similarly to the other cases. The steps per episode also show variability. This behavior suggests that the large weight updates may be overshooting optimal values.

Based on these results, learning rate 0.0005 was selected as the best configuration for subsequent experiments. While 0.0001 achieves marginally better final performance (48-50% vs 45-50%), the training loss analysis reveals that 0.0001 fails to fully optimize the network within the 20000 - episode budget. This suggests the performance gap might simply reflect insufficient training time rather than a fundamental advantage. The 0.0005 configuration provides sufficient stability while learning efficiently, as evidenced by its rapid loss convergence to near-zero. The 0.001 rate is suboptimal because of the final performance deficit (40% vs 45-50%).

B. Batch Size Comparison

Batch size determines how many experiences are sampled from the replay buffer for each gradient update. This parameter affects both the variance of gradient estimates and the frequency of network updates (larger batches require fewer updates for the same number of training steps). Three values were tested: 64 (baseline), 128 (moderate), and 256 (large), all using learning rate 0.0005 and buffer capacity 10000.

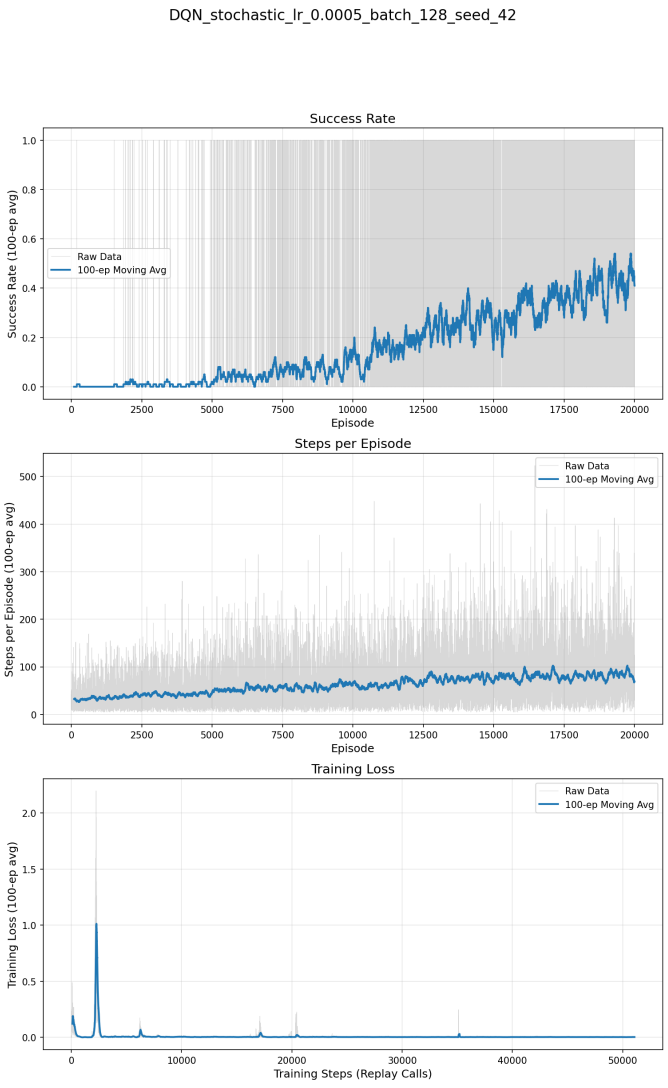

Fig. 15. DQN with batch size 128 (20,000 episodes, lr=0.0005, buffer=10k, seed=42). The training curve shows slower and more unstable learning, reaching approximately **45%** by episode 20,000. This configuration performs notably worse than batch size 64, suggesting that moderate batch sizes may not be optimal for this task.

As shown in Figure 11 (batch size 64, from Section 4.2.1) and Figures 15 and 16, batch size has a non-monotonic effect on performance. As seen, the baseline configuration with batch size 64 achieves the best result, reaching approximately 60% success rate.

Doubling the batch size to 128 produces a degradation in performance. The final success rate drops to 45%, the learning

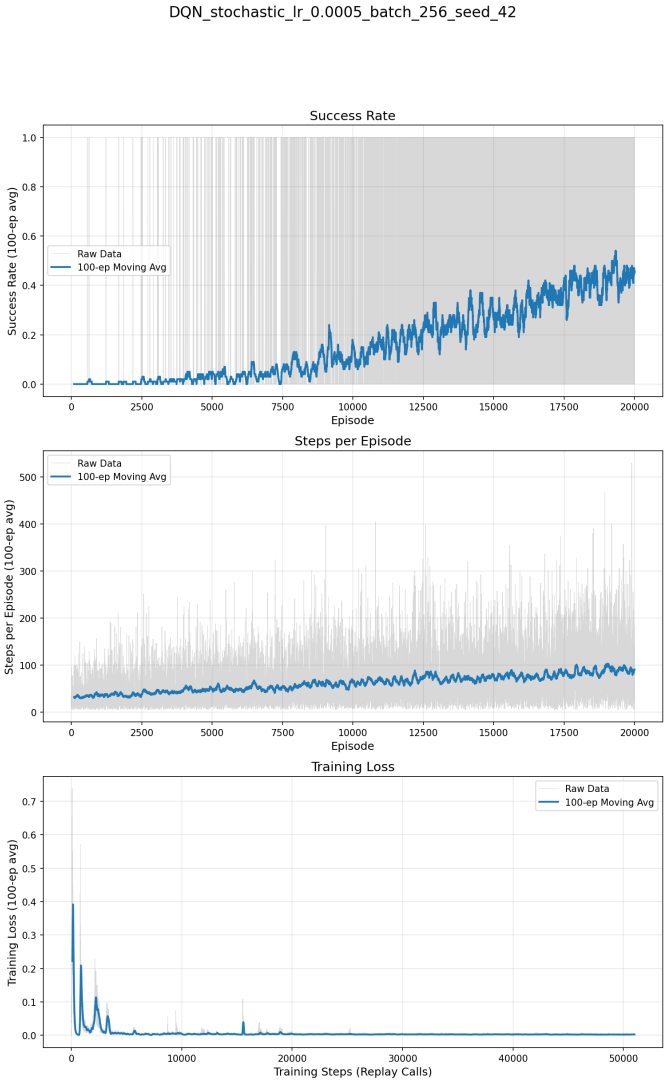

Fig. 16. DQN with batch size 256 (20,000 episodes, lr=0.0005, buffer=10k, seed=42). The training curve shows similar performance compared to batch size 128, also reaching **45%**. However, this still falls short of the batch size 64 configuration.

appears slower and less efficient. This result is counterintuitive, as larger batch sizes typically provide more accurate gradient estimates by averaging over more samples. However, in this case, the benefit of reduced gradient variance is outweighed by other factors.

Increasing the batch size further to 256 changes almost nothing, reaching 45% success rate like 128. It still falls short of the 64 configuration. The training curve for batch size 256 is smoother than 128, the larger batch may provides more stable gradients.

Training losses graphs are all converging to zero, they all perform well, nothing to add on this.

The non-monotonic relationship between batch size and performance can be explained by the trade-off between gradient stability and update frequency. With batch size 64, the agent performs more frequent gradient updates (approximately 1.56×

more than batch 128, and 4× more than batch 256 for the same number of environment steps). In sparse reward environments like FrozenLake, this higher update frequency may be critical: when a rare successful trajectory is discovered and added to the replay buffer, the network can begin incorporating this information after just 64 samples rather than waiting to accumulate 128 or 256 samples for larger batches. This rapid incorporation allows the agent to adjust its policy more quickly when positive signals emerge.

Larger batch sizes, while providing smoother gradients through better averaging, reduce update frequency significantly. The results suggest that for this task, frequent updates with moderately noisy gradients (batch 64) are more effective than infrequent updates with stable gradients (batch 128, 256).

C. Replay Buffer Size Comparison

The replay buffer stores past experiences for training, with capacity determining how many transitions are retained before old experiences are overwritten. Two capacities were tested: 10000 (baseline) and 50000 (large), both using learning rate 0.0005 and batch size 64.

As shown in Figure 11 (buffer size 10000, from Section 4.2.1) and Figure 17, the smaller buffer (10000) outperforms the larger buffer (50000), achieving 60% versus 50-55% final success rate. This result is counterintuitive, as conventional wisdom suggests that larger replay buffers provide greater experience diversity and reduce overfitting to recent experiences. However, in this specific task, the smaller buffer proves more effective.

The training curve for the 10k buffer, as seen, shows consistent improvement throughout all 20000 episodes, with the success rate climbing from near-zero to 60%. The 50k buffer, while also showing improvement, exhibits a slower learning trajectory and plateaus at a bit lower final performance.

This performance gap, though modest (60% vs 50-55%), can be explained by the non-stationarity of the learning process. As the DQN agent's policy evolves during training, the distribution of collected experiences changes. The 50000-capacity buffer retains experiences for much longer, meaning that early, low-quality experiences from the initial random exploration phase persist into later training stages. When the agent samples mini-batches, these outdated experiences dilute the learning signal from more recent, higher-quality trajectories generated by the improved policy.

In the sparse reward setting of FrozenLake, where successful episodes remain rare even for the improved policy, this dilution is particularly problematic: the larger buffer contains proportionally more failed attempts from early training, reducing the frequency with which the network learns from successful experiences. The 10000-capacity buffer, by overwriting older experiences more quickly, maintains training data more representative of the agent's current capability and a higher proportion of recent successful trajectories, resulting in slightly more effective learning. Based on these results, the 10000-capacity buffer was selected for subsequent experiments, demonstrating that larger replay buffers are not always advantageous — particularly in non-stationary sparse reward environments where policy improvement changes the quality of collected data over time.

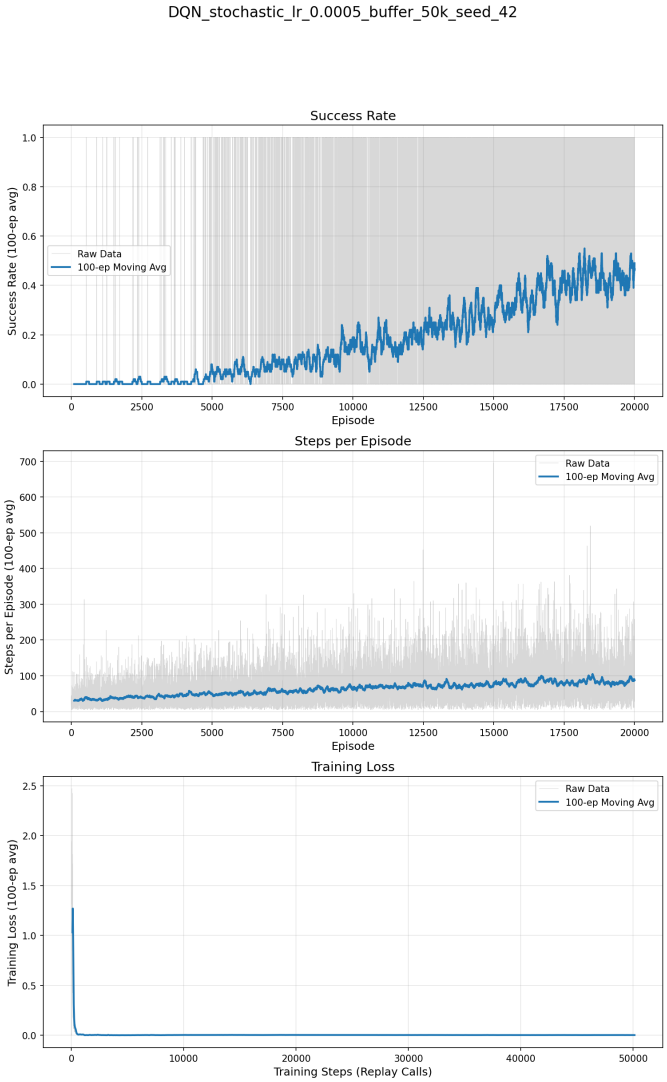

Fig. 17. DQN with buffer size 50,000 (20,000 episodes, lr=0.0005, batch=64, seed=42). The training curve reaches approximately **50%** by episode 20,000. The larger buffer underperforms compared to the 10k configuration, indicating that increased buffer capacity does not improve learning for this task.

Synthesis and Optimal Configuration Selection

Table IV summarizes the results of all hyperparameter experiments conducted in this section. The best configuration identified is: learning rate 0.0005, batch size 64, and buffer capacity 10000.

TABLE IV
HYPERPARAMETER SENSITIVITY ANALYSIS SUMMARY (ALL EXPERIMENTS ON STOCHASTIC FROZENLAKE-V1, SEED=42, 20,000 EPISODES)

Configuration	Success Rate (%)
Learning Rate (batch=64, buffer=10k)	
LR = 0.0001	48-50
LR = 0.0005 ✓	45-50
LR = 0.001	40
Batch Size (lr=0.0005, buffer=10k)	
Batch = 64 ✓	60
Batch = 128	45
Batch = 256	45
Buffer Size (lr=0.0005, batch=64)	
Buffer = 10k ✓	60
Buffer = 50k	50-55

The experiments reveal three key findings. Learning rate 0.0005 provides the best balance between convergence speed and stability, with training loss converging effectively to near-zero. Batch size exhibits a non-monotonic relationship with performance: the smallest tested value (64) outperforms both intermediate (128) and large (256) configurations, demonstrating that frequent updates with moderately noisy gradients are more effective than infrequent updates in sparse reward environments. Counterintuitively, the smaller replay buffer (10k) outperforms the larger one (50k) by maintaining more policy-relevant experiences, highlighting the importance of experience freshness in non-stationary learning settings. This optimal configuration will be used for the multi-seed robustness analysis in Section 4.2.3.

Key Takeaway: systematic hyperparameter analysis establishes the choosen DQN configuration as learning rate 0.0005, batch size 64, and buffer capacity 10000. Notably, both batch size and buffer size exhibit strange patterns: smaller values outperform larger ones, demonstrating that frequent updates and faster experience turnover are critical for learning in sparse reward, non-stationary environments.

3) Multi-Seed Robustness Analysis: to obtain a rigorous assessment of the DQN agent's performance on the stochastic FrozenLake task, a final suite of experiments was conducted using three independent random seeds (42, 123, 999). Each experiment used the optimal configuration identified in Section 4.2.2: learning rate 0.0005, batch size 64, buffer capacity 10000, and 20000 training episodes. Individual results are shown in Figures 11, 18, and 19, with aggregated statistics summarized in Table V.

As shown in Figure 11 (seed 42, from Section 4.2.1) and Figures 18 and 19, the training runs exhibit different trajectories but all demonstrate substantial learning. Seed 42 produces the best outcome with 60% training success rate.

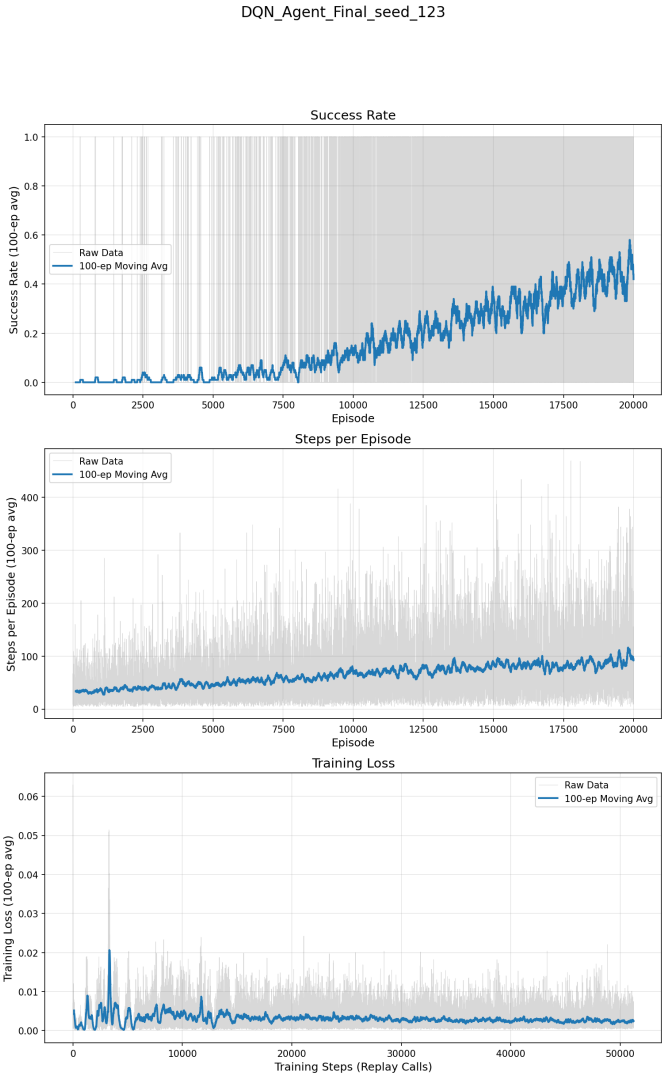

Fig. 18. DQN performance with seed=123 (20000 episodes, optimal configuration). Training curve shows steady improvement, reaching around **55%** by episode 20000. The agent demonstrates consistent learning throughout training.

TABLE V
MULTI-SEED PERFORMANCE SUMMARY FOR DQN AGENT (STOCHASTIC FROZENLAKE-V1, 20000 EPISODES, TRAINING PERFORMANCE)

Seed	Success Rate (%)	Avg Steps
42	60	117
123	55	118
999	42	91
Mean ± Std	**52.3 ± 9.3**	**108 ± 15.3**

This represents strong performance in the stochastic environment, successfully reaching the goal in 6 out of every 10 episodes.

Seed 123 achieves 55% success rate, 5 percentage points lower than seed 42. The average of approximately 118 steps matches seed 42, indicating that when successful, this agent

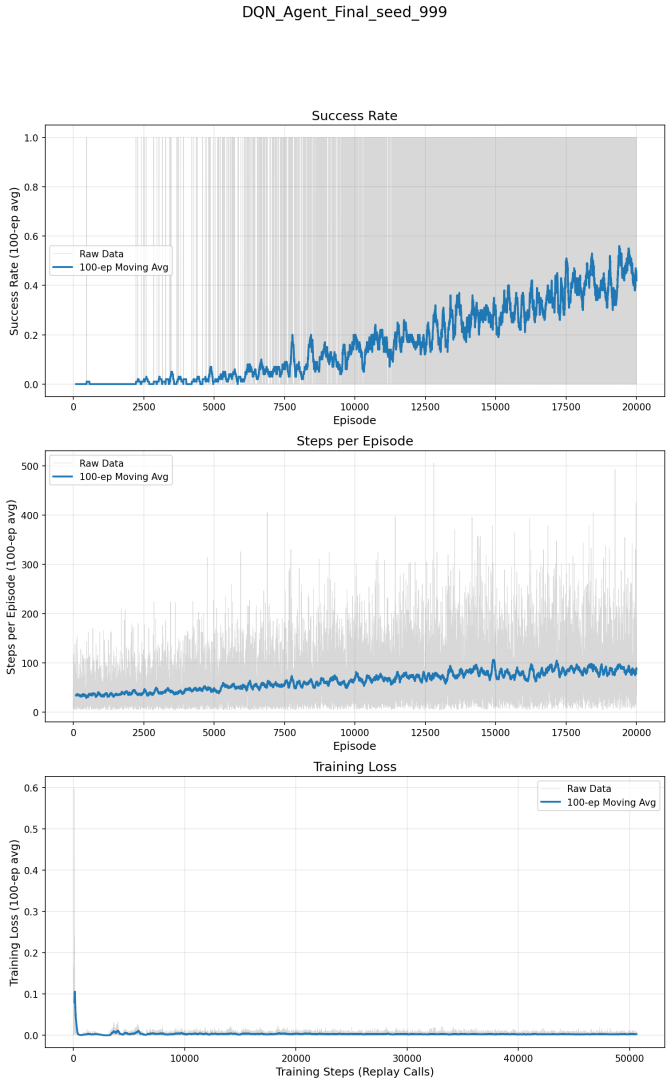

Fig. 19. DQN performance with seed=999 (20000 episodes, optimal configuration). Training curve reaches approximately **40-45%** by episode 20000, representing the weakest performance among the three seeds tested.

navigates with similar efficiency. The performance gap reflects slightly lower reliability rather than less efficient pathfinding.

Seed 999 represents the weakest outcome with around 42% success rate. The training curve sets around 40-45%, lower than the other seeds. This agent takes fewer steps per episode (91) than the others. This shorter path length reflects more frequent early termination: when the agent fails, it tends to fall into holes closer to the starting position, reducing the average episode length.

The aggregated statistics in Table V show the DQN agent's capabilities in the stochastic environment. The mean success rate across three seeds is 52.3 ± 9.3, indicating that the agent successfully reaches the goal in approximately half of all episodes. The standard deviation of 9.3 percentage points represents a variation of 18%, showing moderate consistency across initializations. Although seed 42 (60%) performs 1.43× better than seed 999 (42%), all three seeds achieve success rates above 40% — substantially higher than the Tabular agent.

Key Takeaway: the DQN agent achieves discrete results across the three seeds, representing an important improvement over the Tabular agent.

4) Qualitative Analysis: Visualizing the Learned Policy: while the quantitative results demonstrate the DQN agent's nice performance, examining the learned policy itself provides insight into how the neural network has solved the navigation problem. Figure 20 **visualizes** the greedy policy ($\epsilon = 0$) learned by the best-performing agent (seed 42), showing the action with highest Q-value for each non-terminal state.

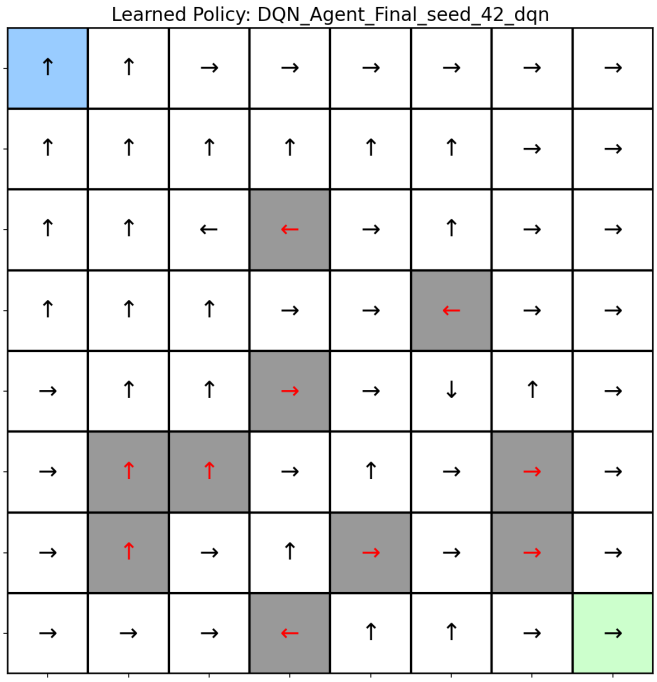

Fig. 20. Learned policy visualization for DQN agent (seed=42). Arrows indicate the greedy action (highest Q-value) for each state. Blue cell: start position. Green cell: goal position. Gray cells: holes (terminal states). Red arrows: actions hole states.

At first glance, the policy appears counter-intuitive. The goal is located in the bottom-right corner (row 8, column 8), yet many arrows point upward or leftward—directions that seem to move away from the target. Some actions near holes even point directly toward those hazards. This raises a natural question: how can an agent achieving 99% evaluation success rate (remind: this 99% is the greedy policy performance) exhibit such apparently suboptimal behavior?

The explanation lies in recognizing that this policy was optimized for a stochastic environment, not a deterministic one. In the deterministic setting, the optimal policy would simply point directly toward the goal from each state, following the shortest geometric path. However, with `is_slippery=True`, each intended action succeeds only 1/3 of the time, with the remaining 2/3 probability split between perpendicular movements. The learned policy reflects an adaptation to this uncertainty, prioritizing statistical robustness.

Perhaps the most revealing pattern is the agent's use of the grid boundaries as **"safety rails"**. The two leftmost columns exhibits a strong preference for upward movement. This likely reflects a learned strategy: when positioned against the left wall and choosing UP, the possible slip outcomes are left (blocked by the boundary), up (intended), or right. By selecting this action, the agent effectively eliminates one of the three possible directions — the boundary prevents leftward slips. This reduces unpredictability and channels movement into more controllable patterns. Similarly, the concentration of upward arrows throughout the grid suggests the agent may be using the top boundary as a navigation aid, reaching the upper edge where further upward slips are impossible, then traversing horizontally toward the goal with reduced vertical uncertainty.

The arrows pointing toward holes provide further evidence of sophisticated probabilistic reasoning. Consider a state adjacent to a hole where the policy selects the action that would deterministically lead into that hazard. This appears suicidal in a deterministic world, but in the stochastic setting, it represents a **calculated risk**. With only 1/3 probability of moving in the intended direction, the agent has 2/3 probability of slipping perpendicular to the chosen action. Apparently, the neural network has determined that the Q-values of states reachable via these perpendicular slips are sufficiently high to justify accepting the 1/3 risk of falling into the hole. The agent is making probabilistic bets based on learned value estimates across multiple possible outcomes.

This policy can be called as an **"alien intelligence"** optimized for a world fundamentally different from human intuition. Humans naturally think in terms of deterministic paths and geometric shortcuts. The DQN has learned to think in terms of probability distributions over successor states, value expectations across stochastic outcomes, and risk-reward tradeoffs that account for slip dynamics. The policy does not encode the shortest path to the goal; it encodes the statistically most robust trajectory through a space of uncertain transitions. Each action represents not just a movement direction but a carefully weighted gamble over three possible outcomes, with the weights learned through thousands of episodes of trial and error.

Key Takeaway: the learned policy visualization reveals that the DQN agent has developed a sophisticated navigation strategy adapted to environmental stochasticity. The policy exhibits counter-intuitive behaviours such as moving away from the goal, using grid boundaries as safety mechanisms, and taking apparently risky actions near holes. These patterns reflect probabilistic reasoning over stochastic state transitions, demonstrating that the neural network has learned to optimize for statistical robustness rather than deterministic path length.

C. Comparative Analysis

This section directly compares the two agents performance on the stochastic FrozenLake task. Figure 21 visualizes the learning trajectories across the three seeds seen, while Table VI summarizes the key performance metrics.

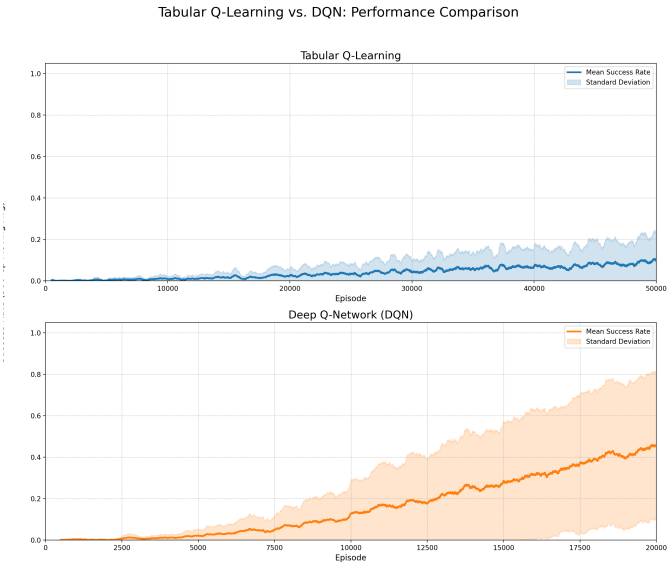

Fig. 21. Learning curves comparison between Tabular Q-Learning (top) and DQN (bottom). Each curve shows the mean success rate across three seeds, with shaded regions indicating standard deviation. The Tabular agent plateaus around 12% after 50000 episodes, while the DQN agent reaches approximately 45% after 20000 episodes.

TABLE VI
FINAL PERFORMANCE COMPARISON: TABULAR Q-LEARNING VS DQN (STOCHASTIC FROZENLAKE-V1, MULTI-SEED ANALYSIS, TRAINING METRICS)

Metric	Tabular	DQN	Improvement
Mean Success Rate	12.1% ± 8.0%	52.3% ± 9.4%	4.3×
Coefficient of Variation	66%	18%	3.7× lower
Best Seed Performance	21%	60%	2.9×
Worst Seed Performance	5.4%	42%	7.8×

As shown in Figure 21, the performance gap between the two methods is visually striking. The Tabular agent's learning curve (top panel) exhibits minimal progress, climbing slowly from near-zero to approximately 10-12% over 50000 episodes. The curve remains relatively flat throughout training, suggesting the agent has reached the limits of what tabular representation can achieve in this stochastic setting. The DQN agent's learning curve (bottom panel) tells a different story: uniform, consistent improvement from episode zero through 20000, reaching approximately 45-50% by the end of training. The slope of the DQN curve indicates active learning throughout the entire training period, with no evidence of premature convergence.

The quantitative comparison in Table VI confirms this gap. The DQN agent achieves 52.3% mean success rate across three seeds, representing a 4.3× improvement over the Tabular agent's 12.1%. This translates to a practical difference of succeeding in approximately half of all episodes versus succeeding in roughly one out of every ten. Even more telling is the comparison of worst-case scenarios: the DQN's weakest seed (42%) still outperforms the Tabular agent's best seed

(21%) by 2×, massive difference.

Beyond absolute performance, the agents exhibit different levels of robustness to initialization. The coefficient of variation (CoV), defined as the ratio of standard deviation to mean, quantifies the relative variability of performance across seeds. The Tabular agent's CoV of 66% indicates that the standard deviation (8.0%) is two-thirds as large as the mean performance (12.1%). This high relative variance means that performance is critically dependent on random seed — essentially a lottery where initialization determines what percentage of success the agent achieves. The DQN agent's CoV of 18% represents a 3.7× reduction in relative variability. Some seed dependence remains (standard deviation of 9.4 percentage points on a mean of 52.3%), but the proportional impact is much smaller. The neural network's generalization capability appears to provide a form of robustness: even when early exploration is less fortunate, the agent can still extract useful patterns from varied experiences and converge to reasonable performance.

The fundamental explanation for this gap lies in the difference between memorization and generalization. The Tabular agent maintains explicit Q-values for each of the 256 state-action pairs. In the deterministic environment, this direct representation can be advantageous — once a successful path is discovered, it can be encoded precisely. On the contrary, in the stochastic environment, the noisy transitions prevent accurate value estimation. Each state-action pair must be visited many times to average out the randomness, and even then, the learned values are specific to those exact states. The agent cannot leverage knowledge from one region of the state space to inform decisions in another region, even if the underlying patterns are similar.

The DQN agent, by contrast, uses a neural network to approximate the Q-function. This function approximation inherently performs generalization: the network learns features and patterns that apply across multiple states. When the agent experiences a successful trajectory, the network updates not just the Q-values for the specific states visited but also the representations that underlie those values. States with similar features will have their Q-values adjusted accordingly, allowing knowledge transfer across the state space. In the stochastic setting, this generalization is particularly valuable.

The environment's randomness naturally generates diverse experiences, the same intended action sequence produces many different actual trajectories. Rather than treating each trajectory as an independent data point (as the Tabular method does), the neural network **extracts common patterns**: "states near holes are dangerous", "moving toward the goal region increases expected return", "using walls as boundaries reduces unpredictability". These learned features enable the agent to make informed decisions even in states that were rarely or never visited during training, and to maintain reasonable performance across different random seeds.

Key Takeaway: the DQN agent outperforms Tabular Q-Learning, showing the power of function approximation in stochastic sparse reward environments. Beyond performance, DQN exhibits 3.7× lower relative variance across seeds (CoV: 18% vs 66%), indicating greater robustness to initialization. Even DQN's worst seed outperforms Tabular's best seed by 2×. Overall, the comparison displayed validates the hypothesis that function approximation is essential for achieving robust performance in non-deterministic environments with sparse rewards.

V. Discussion

The experimental results presented in Section 4 demonstrate a clear performance advantage for Deep Q-Networks over Tabular Q-Learning in the stochastic FrozenLake environment. However, the value of this comparative study extends beyond the raw numbers. This section reflects on the broader implications of the findings, examining trade-offs between the two approaches and highlighting unexpected observations that emerged during experimentation.

A. Q-Learning vs DQN: The Trade-offs

The stark performance difference between Tabular Q-Learning and DQN might suggest that tabular methods are obsolete. This conclusion may be premature. Each approach offers distinct advantages depending on the problem characteristics.

Tabular Q-Learning excels in small, fully observable environments where exhaustive state-action exploration is feasible. The method provides theoretical guarantees: under appropriate conditions (infinite exploration of all state-action pairs and suitable learning rate decay schedules), tabular Q-learning is guaranteed to converge to the optimal Q-function in finite MDPs. The learned Q-table is fully interpretable — each entry has a clear, independent meaning. Training may be fast, since it requires no gradient computation and updates are computationally inexpensive (simple arithmetic operations). For problems with tens or hundreds of states, these properties make tabular methods the practical choice.

The FrozenLake 8×8 environment sits at an interesting boundary. With 64 states, the problem appears small enough for tabular representation. Yet the results reveal a critical insight: state space size alone **does not** determine method suitability. The combination of stochasticity and sparse rewards creates sample efficiency challenge that amplifies the limitations of tabular learning. Furthermore, successful trajectories are rare, meaning most state-action pairs receive zero reward during training. The stochastic transitions inject noise into every Bellman update, making value estimates unreliable unless each state-action pair is visited many times. In effect, the sparse reward structure enlarges the "effective" state space by reducing the density of useful training signal. The tabular agent cannot leverage similarities between states to compensate for this data scarcity — each entry in the Q-table must be learned independently through repeated direct experience.

DQN addresses this limitation through function approximation, but introduces its own complexities. The neural network architecture, learning rate, batch size, and replay buffer capacity all require careful tuning. As showed in Section

4.2.2, suboptimal choices can significantly degrade performance. Moreover, training is computationally more expensive, requiring gradient backpropagation through the network for each update, the learned policy is less interpretable — even if the action selections can be visualized (Section 4.2.4), understanding why the network makes specific choices requires analysing learned features, which is non-trivial. Perhaps most importantly, DQN lacks the strong convergence guarantees of tabular methods. Function approximation can introduce instability, and there is no assurance that training will converge to an optimal or even stable solution.

The trade-off, then, is between simplicity with limited generalization versus complexity with powerful abstraction. For this particular task — a moderately sized state space with significant stochasticity and sparse rewards — the generalization capability of DQN proves essential. The neural network's ability to extract patterns across similar states enables learning from limited positive signal. However, variations of this problem could cause trade-off shifts. Increasing the density of rewards, reducing stochasticity or further shrinking the state space might tip the balance back toward tabular methods. The lesson is not that one approach dominates universally, but rather that **environmental characteristics** determine which method is appropriate.

B. Unexpected Findings and Their Implications

Several observations during experimentation challenged possible initial expectations and warrant deeper reflection.

The "golden seed" phenomenon — where seed 42 achieved 60% training success for DQN - raises questions about the nature of initialization sensitivity in deep reinforcement learning. While DQN exhibits lower relative variance than Tabular Q-Learning, some **seed dependence** clearly remains. This suggests that even with function approximation, the early exploration phase matters. Perhaps certain initializations lead the network to discover features or representations that facilitate faster learning, creating a form of "momentum" that persists throughout training. The existence of such advantageous initializations implies that practitioners should not rely on single-seed results, even when using methods expected to be robust. The common practice of reporting only the best run from several attempts may systematically overestimate true expected performance.

The non-monotonic relationship between batch size and performance is particularly surprising. Conventional lore suggests that larger batches provide more accurate gradient estimates by averaging over more samples, which should improve stability and convergence. Yet batch size 128 performed worse than both 64 and 256. One possible explanation relates to the interaction between batch size and update frequency in sparse reward settings. With batch 64, the agent updates the network more frequently, allowing it to react quickly when rare successful experiences appear in the replay buffer. Batch 128 may represent an unfortunate middle ground: large enough to significantly reduce update frequency (compared to 64), but not large enough to provide the gradient quality benefits of very large batches. This finding suggests that the best batch size may depend critically on reward density, a consideration not always emphasized in deep RL literature, which often focuses on batch size's effect on gradient variance and computational efficiency.

The counter-intuitive learned policy visualization (Section 4.2.4) provides perhaps the **most thought-provoking result**. Actions that appear suboptimal or even trivial in a deterministic world — moving away from the goal, using boundaries as safety mechanisms, even pointing toward holes — reveal smart probabilistic reasoning when viewed through the lens of stochastic transitions. This highlights a fundamental challenge in interpretable AI: a policy optimized for an uncertain environment may appear illogical when evaluated by deterministic human intuition. The policy does not encode "go to the goal"; it encodes "maximize expected return given slip probabilities". As reinforcement learning is increasingly applied to real-world problems with inherent uncertainty — robotics with imperfect actuators, autonomous vehicles with unpredictable road conditions, financial trading with market volatility — better frameworks should be developed for understanding and validating policies that optimize for stochastic objectives rather than deterministic plans.

VI. Conclusion

A. Summary of Key Findings

This study compared Tabular Q-Learning and Deep Q-Network agents on the FrozenLake-v1 8×8 stochastic navigation task, providing empirical evidence for the advantages of function approximation in environments characterized by sparse rewards and stochastic state transitions.

The Tabular Q-Learning agent achieved a mean success rate of ~12% across three independent seeds after 50000 training episodes, with high sensitivity to initialization (coefficient of variation: 66%). The agent showed limitations in extracting signal from noisy transitions, with performance plateauing early in training and exhibiting minimal improvement despite extended exploration. The inability to generalize across similar states resulted in sample-inefficient learning, where each state-action pair required independent discovery and repeated visitation to achieve reliable value estimates.

The DQN agent achieved a mean success rate of ~52% across three seeds after 20000 training episodes, representing around a 4x improvement in performance and reduction in relative variance (coefficient of variation: 18%). The neural network's function approximation capability enabled knowledge transfer across the state space, allowing the agent to learn robust navigation strategies from limited successful experiences.

Qualitative analysis of the learned policy demonstrated that the DQN agent developed strategies adapted to stochastic dynamics rather than geometric optimality. Actions that appear counter-intuitive in deterministic settings reflect probabilistic reasoning over slip outcomes and risk-reward trade-offs.

The comparative analysis also established that function approximation becomes essential when stochasticity and sparse rewards create sample efficiency challenges that exceed the practical limits of exhaustive state-action exploration.

B. Potential Future Work

Several directions could extend this investigation and address limitations of the current study.

Scaling to larger state spaces would test whether the performance advantage of function approximation increases with problem complexity. The FrozenLake environment could be extended to 16×16 or 32×32 grids, increasing the state space to 256 or 1024 states respectively. Such experiments would reveal whether tabular methods become entirely impractical beyond a certain threshold, and whether DQN's generalization capability provides increasing benefits as the state space grows. Additionally, testing on environments with continuous state spaces would eliminate the discretization that makes tabular methods feasible at all, providing a clearer assessment of deep RL's advantages in realistic settings.

Investigating alternative deep RL algorithms would establish whether the observed advantages are specific to DQN or general properties of function approximation. For instance, algorithms such as Double DQN can be tested to compare these methods on the same stochastic FrozenLake task.

Introducing multiple agents in a shared environment would add a layer of complexity absent from the current single-agent setting. Multi-agent variants of FrozenLake could require cooperation (agents must coordinate to reach a shared goal) or competition (agents race to reach the goal first). Such scenarios would test whether function approximation methods can learn effective coordination or competitive strategies, and whether the communication and credit assignment challenges of multi-agent RL favor certain algorithmic approaches.

Finally, transfer learning experiments could assess whether policies learned on FrozenLake generalize to related but distinct navigation tasks. Training an agent on the 8×8 stochastic environment and evaluating it on different grid sizes, hole configurations or reward structures would test the robustness of learned representations. Such experiments would be particularly relevant for real-world applications where the deployment environment may differ from the training environment.

References

[1] Gymnasium Documentation https://gymnasium.farama.org/

[2] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[3] Anthropic, "Claude 4.5 Sonnet," https://www.anthropic.com/claude

[4] Google DeepMind, "Gemini 2.5 Pro," https://deepmind.google/technologies/gemini/

[5] Original Paper on DQN https://arxiv.org/pdf/1312.5602

[6] Course material: "Reinforcement Learning," Lecture slides by prof. Fabio Patrizi

[7] "Deep Q-Network Architecture Diagram," *WikiDocs* https://wikidocs.net/174548

[8] "OpenAI Gym Environments Used as Experiment Tasks," *ResearchGate* https://www.researchgate.net/figure/OpenAI-gym-environments