Deep Reinforcement Learning and GridWorlds

Nicholas Lai Jin Yung Olufemi Davies Ahmed Idris

Code: Frozen Lake, MiniGrid and Deep RL

1 The Frozen Lake

1.1 Policy Iteration vs Value Iteration Convergence

Policy iteration is expected to converge faster than value iteration because the former improves the policy directly, whereas the latter updates value estimates cell by cell before convergence.

Results:

- Policy Iteration: **6 iterations**
- Value Iteration: **60 iterations**

1.2 Model-Free Algorithm Learning Curves

The plot shows that tabular and linear methods (Sarsa, Q-learning) gradually improve and stabilise around a return of 0.4–0.5. The Deep Q-Network learns rapidly initially (around episode 500) but exhibits significantly higher variance and instability compared to the tabular methods.

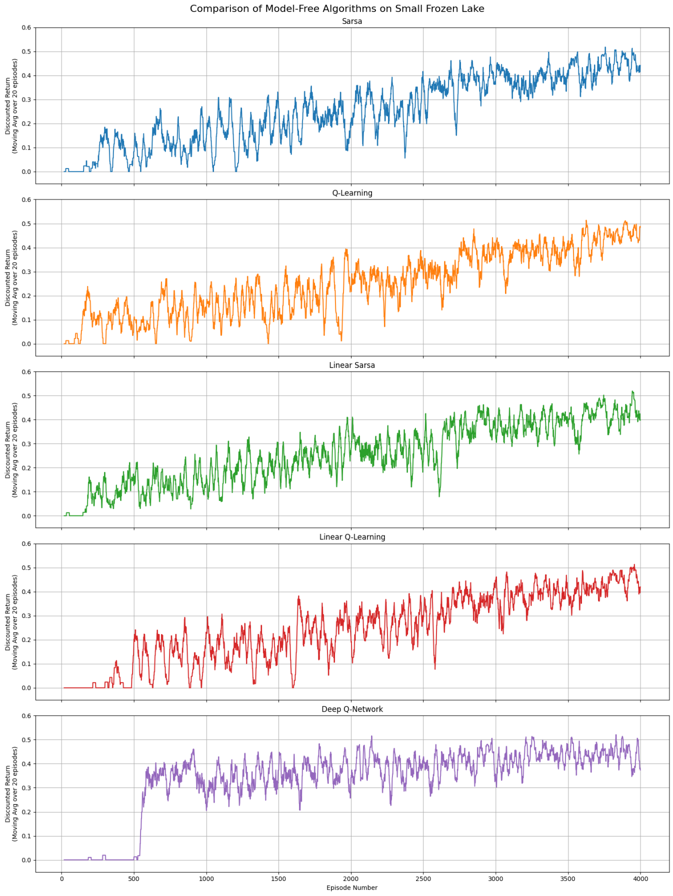

Figure 1

Total Discounted Returns (Sum over all episodes):

- Sarsa: 1003.42
- Q-Learning: 992.58
- Linear Sarsa: 1022.72
- Linear Q-Learning: 968.87
- Deep Q-Network: 1339.49

1.3 Parameter Tuning for Convergence

1.3.1 Small Frozen Lake

A grid search was conducted over learning rates (η) and exploration factors (ϵ) to minimise convergence time. A maximum episode cap of 10,000 was employed.

Sarsa: Minimum episodes $\approx$ 3,278. Optimal parameters: $\eta = 0.5$, $\epsilon = 0.3$.

Q-learning: Minimum episodes $\approx$ 1,479. Optimal parameters: $\eta = 0.9$, $\epsilon = 0.9$.

Analysis: Q-learning converged more than twice as fast as Sarsa. High learning rates and high initial exploration allowed Q-learning to propagate the reward signal efficiently in this small environment.

1.3.2 Big Frozen Lake

The grid search for the big frozen lake was conducted with a maximum of 20,000 episodes to account for the larger search space.

Result: Neither Sarsa nor Q-learning converged to the optimal policy within 20,000 episodes for any combination of parameters.

Analysis: The algorithms failed because the big lake has a much larger state space (64 tiles vs. 16) and rewards are sparse (only at the goal). Without eligibility traces or more advanced exploration, single-step tabular methods require significantly more than 20,000 episodes to propagate the reward signal from the goal back to the start state.

1.4 Linear Function Approximation Interpretation

Interpretation of θ: When each state-action pair $Q(s, a)$ is represented by a unique one-hot feature vector $\phi(s, a)$ (where only one element is 1 and the rest are 0), the linear approximation $Q(s, a) = \theta^T \phi(s, a)$ simply selects a single element from θ. Therefore, each element θ_i corresponds directly to the estimated Q-value for a specific state-action pair. The vector θ is effectively a flattened Q-table.

Why Tabular is a Special Case: Tabular algorithms are a special case of linear algorithms because the update rules become mathematically identical under one-hot encoding:

- In linear methods, the update is proportional to the gradient $\nabla_\theta Q(s, a) = \phi(s, a)$.
- With one-hot features, this gradient is a vector with a single 1.
- Consequently, the update $\theta \leftarrow \theta + \alpha\delta\phi(s, a)$ modifies only the single parameter θ_i associated with the current (s, a) pair, exactly matching the tabular update $Q(s, a) \leftarrow Q(s, a) + \alpha\delta$.

1.5 ϵ-Greedy Policy in DQN Training

Using an ϵ-greedy policy is necessary because it promotes exploration, which addresses two key issues in DQN training. First, exploration prevents the agent from prematurely converging to a suboptimal policy by continuing to try alternative actions instead of always exploiting the current (and most likely inaccurate) Q estimates. Second, it ensures broader coverage of the state-action space, so that more states and actions are visited and learned about, rather than the agent focusing only on a small subset of states induced by a greedy policy.

1.6 Target Q-Network Necessity

Mnih et al. [1] argue that using a single neural network for both estimating Q-values and computing the TD target leads to instability when combined with nonlinear function approximation. In standard Q-learning, the target

$$y_t = r_t + \gamma \max_{a'} Q_\theta(s', a')$$

depends on the same parameters θ that are being updated. As a result, each gradient update simultaneously changes both the predicted value $Q_\theta(s_t, a_t)$ and the target y_t, creating a moving target problem. Because the update is bootstrapped, increases in one Q-value can immediately increase the target itself, forming a positive feedback loop that can amplify errors and cause oscillation or divergence.

To address this, DQN introduces a separate target Q-network with parameters θ^- that are held fixed for many updates and only periodically copied from the online network. Targets are computed using this frozen network, making them approximately stationary over many updates. This breaks the harmful feedback loop and significantly stabilises learning.

2 Beyond the Frozen Lake

2.1 Environment

We selected MiniGrid [2] for its flexibility in defining gridworld environments with varying complexity. MiniGrid provides a natural progression from simple navigation tasks (Empty-5x5, Empty-8x8) through object interaction (DoorKey-5x5) to multi-step planning (KeyCorridor, MultiRoom). This allows systematic investigation of how algorithm performance scales with environment complexity.

The state space is partially observable—agents perceive only a 7×7 grid centred on their position. We use the `ImgObsWrapper` to provide image-based observations ($7 \times 7 \times 3 = 147$ dimensions) flattened for input to a multilayer perceptron. The action space is discrete with seven actions: turn left, turn right, move forward, pickup, drop, toggle, and done.

Rewards are sparse: $1 - 0.9 \times (\text{step\_count}/\text{max\_steps})$ upon reaching the goal (maximum ≈ 0.96), and 0 otherwise.

2.2 Algorithms

We compare three algorithms representing different paradigms in deep reinforcement learning:

Algorithm	Type	Key Characteristics
DQN	Off-policy, value	Replay, target networks
A2C	On-policy, actor-critic	Sync. advantage estimation
PPO	On-policy, actor-critic	Clipped surrogate objective

Table 1: Algorithm comparison.

All implementations use Stable Baselines3 with default hyperparameters and `MlpPolicy` networks. We use MlpPolicy with flattened observations rather than CnnPolicy, as the encoded categorical representation differs from pixel-based inputs where CNNs typically excel.

Parameter	PPO	A2C	DQN
Learning rate	3e-4	7e-4	1e-4
Discount (γ)	0.99	0.99	0.99
Batch size	64	–	32
Steps/update	2048	5	–
GAE λ	0.95	1.0	–
Replay buffer	–	–	10^4
Target update	–	–	10k

Table 2: Algorithm hyperparameters (SB3 defaults).

2.3 Experimental Protocol

Each experiment trains for a minimum of 50,000 time steps using Stable Baselines3's default hyperparameters. We vary time step allocations across experiments to balance result quality with computational constraints, extending training to 150k time steps where beneficial—such as on the DoorKey 5x5 environment—to demonstrate clearer learning progress. We deliberately use defaults to ensure fair comparison, as algorithm-specific tuning would introduce confounding factors. The environment is wrapped with `ImgObsWrapper` to extract image observations and `Monitor` to enable episode statistics logging. To support reproducibility, the complete implementation and experiment scripts are provided in the GitHub repository linked at the top of this document.

To ensure reproducibility, we control randomness at multiple levels. Each experiment receives a seed passed to both the algorithm (controlling network initialisation and action sampling) and the environment (controlling initial state distribution). Experiments were run across 1—5 seeds to capture variance inherent to deep RL training; results are reported as mean ± standard deviation. Experiment configurations and raw results are saved to JSON files, enabling exact reproduction. We report training time steps rather than wall-clock time, as time steps are hardware-independent and ensure reproducibility across different computational environments [3].

During training, a custom callback logs episode rewards, lengths, and time steps whenever an episode completes. After training, we evaluate over 10 episodes using deterministic action selection to assess the learned policy without exploration noise. Learning curves are smoothed using a rolling average (window size 50) to reduce noise while preserving the underlying trajectory.

Our analysis focuses on three metrics: final evaluation reward (maximum ≈ 0.96), sample efficiency (reward vs. time steps), and training stability (whether performance is maintained or collapses).

Protocol Strengths: Statistical rigour through five-seed evaluation with variance reporting, hardware-independent reproducibility via time step based metrics, and fair algorithm comparison through standardised default hyperparameters. The progressive environment difficulty (Empty → DoorKey → DistShift) systematically isolates factors affecting performance.

Limitations: Our exclusive use of MLP policies may disadvantage DQN, which was designed for convolutional architectures. Reliance on default hyperparameters, while fair, may underestimate each algorithm's potential. Computational constraints limited some experiments to three seeds and prevented thorough tuning of intrinsic motivation bonuses.

2.4 Analysis

2.4.1 Baseline Experiments: Empty Environments

We establish baseline performance on MiniGrid-Empty-5x5-v0 and MiniGrid-Empty-8x8-v0, two simple navigation tasks where the agent must reach a goal with no obstacles. These experiments assess two key challenges from Arulkumaran et al. [4]: sample efficiency (how quickly agents learn) and training instability (whether learning reliably converges). **We hypothesize that on empty environments (no obstacles, single goal), on-policy actor–critic methods (A2C, PPO) will achieve near-optimal reward quickly** (within 50k time steps), because random exploration frequently encounters the goal and provides dense enough learning signal.

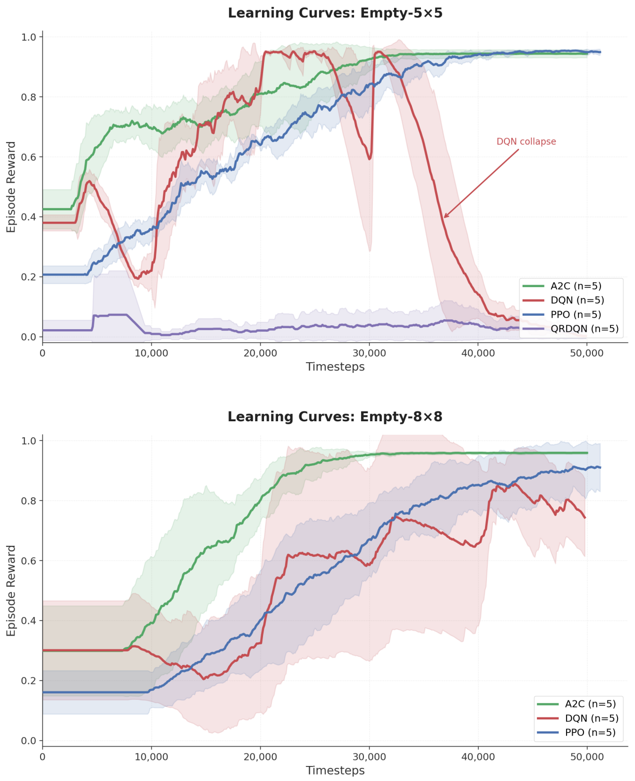

Figure 2: Learning curves for Empty-5x5 (top) and Empty-8x8 (bottom).

Algorithm	Empty-5x5	Empty-8x8
A2C	0.944 ± 0.013	0.959 ± 0.004
PPO	0.955 ± 0.000	0.577 ± 0.471
DQN	0.000 ± 0.000	0.000 ± 0.000

Table 3: Final evaluation reward (mean ± std, 5 seeds). Max ≈ 0.96.

DQN: Systematic Instability. DQN failed completely on both environments. On Empty-5x5, learning curves show DQN reaching 0.85–0.95 within 15,000 time steps before collapsing around 27,000–35,000 time steps. This collapse is systematic: all five seeds exhibit identical failure patterns, ruling out unlucky initialisation.

On Empty-8x8, DQN shows a different failure mode. Training curves suggest partial learning (mean ∼0.7–0.8), yet all seeds evaluate to zero. This discrepancy reveals that DQN temporarily discovers effective policies but fails to retain them—training rewards alone prove misleading.

Both patterns illustrate the training instability challenge identified by Arulkumaran et al. [4]. Experience replay and target networks are insufficient to stabilise DQN when flattened images are processed by an MLP rather than the convolutional architectures DQN was designed for.

A2C demonstrates remarkable consistency: near-optimal performance (0.94–0.96) with minimal variance and 100% success rate on both environments. A2C also converges fastest (∼5,000 time steps on Empty-5x5).

PPO shows unexpected divergence: perfect on Empty-5x5 but high variance on Empty-8x8, with only 3/5 seeds succeeding. This suggests PPO's default hyperparameters may require tuning as state space complexity increases.

2.4.2 Sparse Reward Experiments: DoorKey Environments

We hypothesise that sparse, prerequisite-structured rewards will dramatically reduce sample efficiency compared to dense navigation tasks. DoorKey's multi-step prerequisite structure makes reward signals effectively sparser than Empty environments. Random exploration frequently solves Empty, but almost never discovers the key→door→goal sequence.

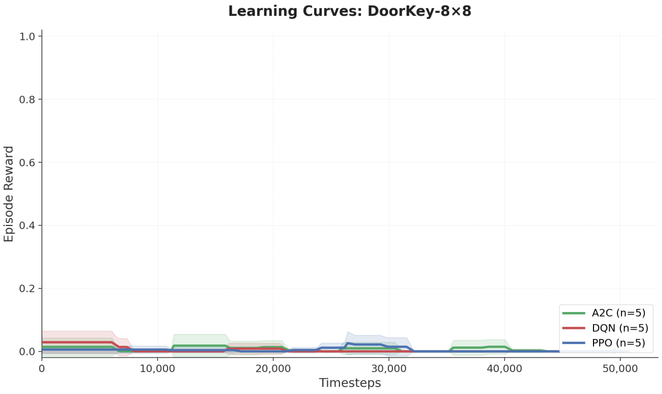

Figure 3: Learning Curves: DoorKey 8x8 (DoorKey-5x5 omitted; identical failure pattern)

50k time steps (5x5 and 8x8): For both DoorKey environments, all algorithms failed completely within 50k time steps: 0% success across 30 runs (3 algorithms × 2 environments × 5 seeds). Even A2C, which excelled on Empty grids, could not discover the key→door→goal sequence. This demonstrates that sample-efficient algorithms on simple tasks become sample-inefficient when exploration is difficult.

150k time steps on 5x5: Increasing training to 150k time steps revealed learning potential. PPO achieved 81% success (4/5 seeds), while A2C showed partial learning (10% mean reward). DQN remained at 0%.

Algo	50k Steps	150k Steps
PPO	0.00 (0/5)	0.81 (4/5)
A2C	0.00 (0/5)	0.10 (1/5)
DQN	0.00 (0/5)	0.00 (0/5)

Table 4: DoorKey-5x5 results.

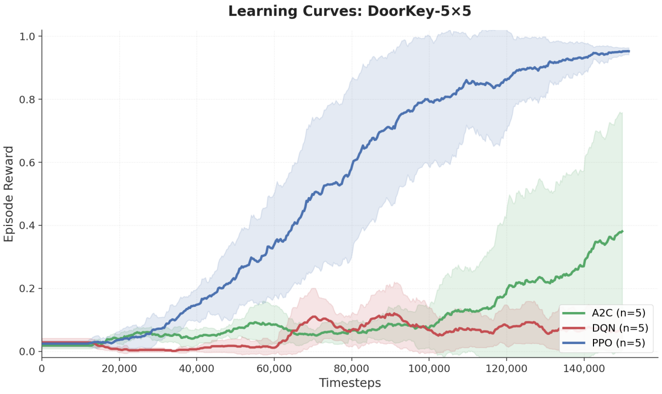

Figure 4: Learning Curves: DoorKey 5x5 with 150k time steps

These results reveal a ∼3× sample efficiency gap: PPO solves Empty-5x5 in ∼20k time steps but requires ∼80k for DoorKey-5x5. The difference stems from reward sparsity—the key→door→goal sequence must be discovered before any learning signal exists.

Algorithm performance proved task-dependent. A2C outperformed PPO on Empty-8x8 (0.96 vs 0.58) but underperformed on DoorKey-5x5 (0.10 vs 0.81). A2C's frequent small updates (n_steps=5) excel with dense rewards but amplify noise when successes are rare. PPO's larger batches (n_steps=2048) average over more experience, providing stable gradients despite sparse feedback.

DoorKey-8x8: DoorKey-8x8 remained unsolvable for PPO at 200k time steps (0/5 seeds). Learning curves in Figure 5 show occasional early spikes when random exploration discovers successful trajectories, but these isolated successes provide insufficient signal. As the policy becomes deterministic, exploration collapses and learning flatlines—a characteristic failure mode in sparse reward settings [5].

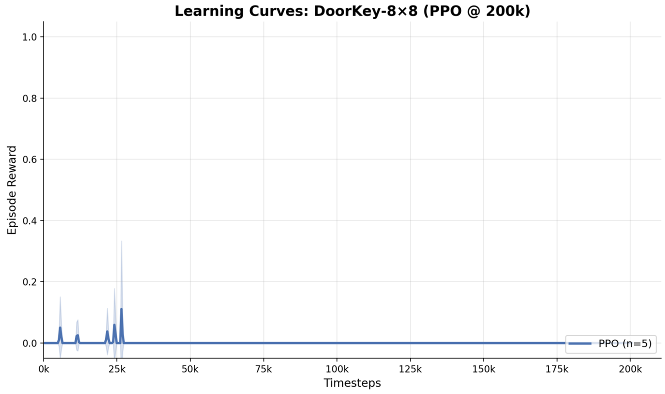

Figure 5: Learning Curves: DoorKey-8x8 (PPO @ 200k)

This scaling of sample requirements with environment size highlights why sample efficiency is a critical challenge in deep RL [4].

2.4.3 Intrinsic Motivation and Reward Shaping

We implemented three intrinsic motivation methods to encourage exploration: count-based bonuses, ICM [6], and RND [7]. Table 5 summarises the hyperparameters used.

Method	Param	Value	Formulation
Count-based	β	0.1	$r_{\text{int}} = \beta/\sqrt{N(s)}$
ICM	β	0.01	$r_{\text{int}} = \beta \cdot \|\hat{s}' - s'\|^2$
RND	β	0.1	$r_{\text{int}} = \beta \cdot \|f(s) - \hat{f}(s)\|^2$

Table 5: Intrinsic motivation hyperparameters. ICM uses $\eta = 0.2$ forward/inverse weighting.

None of these enhancements succeeded on DoorKey-8x8 at 200k time steps, as shown in Table 7. Tuning proved difficult: large bonuses caused agents to prioritise novelty over task completion, while small bonuses provided insufficient exploration incentive.

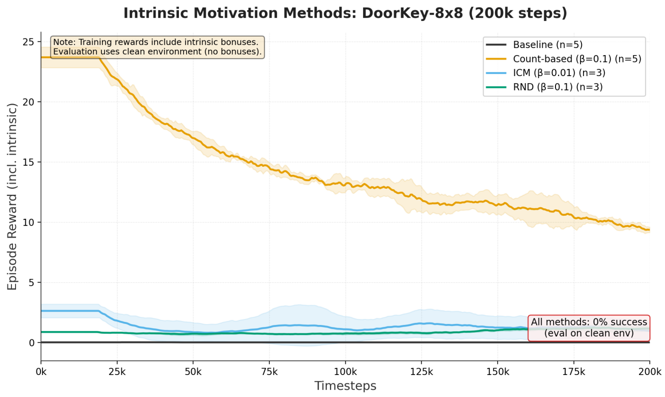

Figure 6: Intrinsic motivation learning curves on DoorKey-8x8. Training rewards include intrinsic bonuses; all methods achieve 0% success when evaluated on clean environment.

Failure mode analysis. Figure 6 reveals why intrinsic motivation methods failed despite high training rewards:

- *Count-based* ($\beta = 0.1$): Training rewards reached 5–25 per episode because bonuses accumulate at every time step. With 640 steps/episode and ~100 unique states, intrinsic reward ≈ 6.4/episode. The agent learned to maximise state coverage rather than task completion.
- *ICM* ($\beta = 0.01$): The forward model quickly learned to predict next states with near-zero error in the deterministic gridworld. Intrinsic rewards diminished after ~50k steps, leaving insufficient signal for exploration.
- *RND* ($\beta = 0.1$): The predictor network rapidly learned the target network's outputs, collapsing the novelty signal. Similar dynamics to ICM: initial exploration spike followed by signal decay.

Reward shaping proved successful. We added intermediate rewards for picking up the key (+0.5) and opening the door (+0.5). At 500k time steps, reward shaping achieved 67% success (2/3 seeds), as shown in Table 6. We extended compute for ICM to 500k for fair comparison; it remained at 0%.

Seed	Key Bonus	Door Bonus	Final Reward
1	+0.5	+0.5	0.578 (success)
2	+0.5	+0.5	0.389 (failure)
3	+0.5	+0.5	0.680 (success)
Mean			0.549 ± 0.119

Table 6: Reward shaping per-seed results on DoorKey-8x8 (500k time steps).

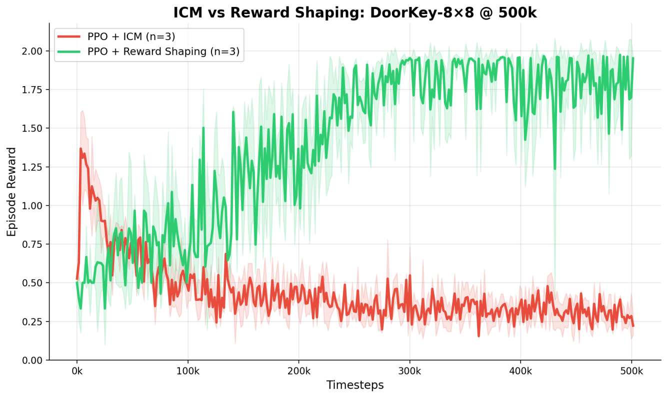

Figure 7: ICM vs reward shaping at 500k time steps. Reward shaping achieves task success while ICM remains at zero.

Method	Steps	Seeds	R	Succ.
PPO Baseline	200k	5	0.00	0/5
A2C Baseline	200k	5	0.00	0/5
DQN Baseline	200k	5	0.00	0/5
PPO + Count	200k	5	0.00	0/5
PPO + ICM	200k	3	0.00	0/3
PPO + RND	200k	3	0.00	0/3
PPO + ICM	500k	3	0.00	0/3
PPO + Shaping	**500k**	**3**	**0.55**	**2/3**

Table 7: DoorKey-8x8 exploration methods comparison. All methods evaluated on clean environment (no intrinsic rewards).

Intrinsic motivation methods, while theoretically appealing, failed to solve hard exploration problems within our compute budget. The core issue is that *task-agnostic* curiosity signals (novelty, prediction error) do not align with *task-specific* objectives—the agent explores broadly but not toward the goal. **Reward shaping, by leveraging domain knowledge to provide intermediate feedback on task-relevant subgoals, proved effective but requires task-specific engineering.** This trade-off between generality and effectiveness remains an open challenge in sparse-reward RL.

This round of experimentation demonstrates that sparse rewards remain a fundamental challenge in deep RL [8]. Even well-performing algorithms require several times more samples on sparse tasks compared to dense-reward equivalents, and the gap widens significantly with environment complexity.

2.4.4 Generalisation Experiments: DistShift Environments

Performing well on unseen environments remains a critical challenge in deep RL [9]. We investigate whether standard algorithms can generalise across minimal environmental variations using MiniGrid's DistShift environments, which share identical objectives but differ only in obstacle placement.

Agents were trained for 100,000 time steps on DistShift1, then evaluated on both DistShift1 (in-distribution) and DistShift2 (out-of-distribution) using deterministic policies. Success was defined as mean reward > 0.5 over 10 episodes, with 5 seeds per configuration.

Algo	DS1	DS2	Train	Test
PPO	0.75 ± 0.38	0.00	4/5	0/5
A2C	0.75 ± 0.38	0.00	4/5	0/5
DQN	0.00	0.00	0/5	0/5

Table 8: Baseline performance on DistShift.

PPO and A2C achieve 80% success on training but 0% on test—complete generalisation failure despite near-identical tasks.

Trajectory analysis revealed the trained policy executes an identical fixed action sequence regardless of environment:

```
DS1: left×3, forward×4, left, forward×6... Goal
(19 steps)
```

```
DS2: Same sequence, walks into lava (step 10)
```

The failure is policy memorisation, not perceptual—the agent observes lava but has learned a state-independent motor program.

Interventions: We tested whether modifying hyperparameters or training data could prevent this memorisation:

Intervention	DS1	DS2	Finding
High entropy (0.1)	0.94	0.00	Still memorised
Small network	0.62	0.00	No effect
Data augment.	0.63	0.00	No effect
CNN + RGB	0.00	0.00	Always "forward"
Procedural	0.00	0.00	Failed to learn
DS1+DS2	1.00	0.20	Two sequences

Table 9: Generalisation intervention results.

Higher entropy improved training performance (0.94 vs 0.62) by encouraging exploration of different trajectories, but the final deterministic policy still collapsed to a fixed sequence—entropy affects *which* sequence is memorised, not *whether* the policy becomes reactive.

MLP policies on small observations learn state-independent action sequences rather than reactive policies. These findings align with Cobbe et al. [9], who showed agents require hundreds of procedurally generated levels for generalisation, and Ghosh et al. [10], who argue that training on limited environments prevents agents from learning which observation features are task-relevant versus coincidental. Robust generalisation requires far more environmental diversity than tested here.

3 Conclusion

This work implemented and evaluated RL algorithms across Frozen Lake (tabular/deep methods) and MiniGrid environments (modern deep RL).

Frozen Lake: Policy iteration converged in 6 iterations versus 60 for value iteration. Q-learning achieved faster convergence than Sarsa (1,479 vs 3,278 episodes). Neither solved the big frozen lake within 20,000 episodes, highlighting sample inefficiency in sparse-reward, large-state-space environments. Linear function approximation with one-hot encoding proved mathematically equivalent to tabular methods.

Beyond Frozen Lake: Our MiniGrid experiments revealed three critical challenges. *Training instability*: DQN exhibited systematic collapse, reaching near-optimal performance before catastrophically forgetting—absent in on-policy methods (A2C, PPO). *Sample efficiency*: sparse prerequisite-structured rewards (DoorKey) required 3× more samples than dense rewards, with DoorKey-8x8 unsolvable at 200k time steps. Intrinsic motivation methods (ICM, RND) failed within our compute budget; reward shaping succeeded but requires domain knowledge. *Generalisation*: agents memorised state-independent action sequences rather than reactive policies, achieving 0% transfer to near-identical test environments.

Limitations: MLP-only policies may have disadvantaged DQN. Default hyperparameters may underestimate algorithm potential, and some experiments were limited to three seeds. **Future work** could address these gaps: Double DQN for stability, hyperparameter sensitivity analysis, procedurally generated environments for generalisation, and model-based methods (e.g., Dreamer) for sample efficiency.

Takeaways: On-policy actor-critic methods demonstrated superior stability compared to off-policy value-based methods in our setting. However, all methods struggled with sparse rewards and generalisation—fundamental open problems in deep RL.

References

[1] Mnih, V., et al. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540), 529–533.

[2] Chevalier-Boisvert, M., et al. (2023). Minigrid & Miniworld: Modular & customizable RL environments. *NeurIPS*.

[3] Machado, M.C., et al. (2018). Revisiting the Arcade Learning Environment. *JAIR*, 61, 523–562.

[4] Arulkumaran, K., et al. (2017). A brief survey of deep reinforcement learning. *IEEE Signal Proc. Mag.*, 34(6), 26–38.

[5] Schaul, T., et al. (2019). Ray interference: A source of plateaus in deep RL. *NeurIPS*.

[6] Pathak, D., et al. (2017). Curiosity-driven exploration by self-supervised prediction. *ICML*.

[7] Burda, Y., et al. (2018). Exploration by random network distillation. *ICLR*.

[8] Fang, Z., et al. (2023). DEIR: Efficient intrinsic motivation via discriminative model. *ICLR*.

[9] Cobbe, K., et al. (2019). Quantifying generalization in reinforcement learning. *ICML*.

[10] Ghosh, D., et al. (2021). Why generalization in RL is difficult: Epistemic POMDPs. *NeurIPS*.