Journal of Applied Artificial Intelligence
https://jaai.sabapub.com
ISSN: 2709-5908

2025 Volume 6, Issue 2: 44–59

DOI: https://doi.org/10.48185/jaai.v6i2.1797

Asynchronous Deep Reinforcement Learning: A Shared Experience Replay Framework

Luckyson Khaidem[1, 1*], Kai Xi[2]

[1] Independent Researcher, Seattle, WA, United States
[2] Independent Researcher, Seattle, WA, United States

Received: 21.09.2025 • Accepted: 22.11.2025 • Published: 30.12.2025 • Final Version: 31.12.2025

Abstract: Off-policy reinforcement learning (RL) algorithms with experience replay have achieved strong performance across a range of decision-making tasks. However, traditional implementations typically rely on a single agent interacting with one environment instance, which can limit exploration diversity and slow convergence. In this paper, we propose an asynchronous multi-agent RL framework that leverages a shared experience replay buffer. Each agent interacts independently with its own environment instance, contributing to a centralized buffer that aggregates diverse trajectories. This setup enhances sample diversity, accelerates learning, and scales efficiently with modern hardware. Our framework is compatible with standard off-policy algorithms such as Double DQN (DDQN) and DDPG, and we demonstrate its effectiveness on a set of representative discrete and continuous control benchmarks. Experimental results indicate that our approach improves learning stability and, in several tasks, reduces time to reach task-specific performance thresholds relative to single-agent baselines. We discuss the theoretical implications of sharing experiences across agents and highlight future extensions.

Keywords: Asynchronous reinforcement learning, Shared experience replay, multi-agent learning, Off-policy algorithms, Deep reinforcement learning

1. Introduction

Reinforcement learning (RL) has emerged as a powerful framework for sequential decision-making, propelled by advances in function approximation via deep neural networks [1]. By casting an agent's interaction with an environment as a Markov Decision Process (MDP), RL seeks to optimize the expected return through trial-and-error learning. While early deep RL successes, such as DQN [1], demonstrated the viability of scaling up neural networks in RL, subsequent innovations have continued to drive performance on increasingly complex tasks [2, 3].

A core concept that has enabled these improvements is experience replay [4], which stores observed transitions in a buffer and then samples them in batches for off-policy updates. Experience replay reduces the correlation between consecutive samples, leading to more stable and sample-efficient learning. Despite these advantages, conventional usage of replay involves a single agent collecting experiences from one environment instance. This fundamentally limits the diversity of data to what a single policy can gather over time, which can result in slower coverage of the environment's state-action space.

To address these limitations, we propose a multi-agent architecture in which each agent interacts with its own copy of the environment, and all agents contribute their experiences to a shared replay buffer. The key intuition behind this design is that multiple agents, especially when they differ in

[1] khaidem90@gmail.com

initialization, policy parameters, or random seeds, are likely to explore different parts of the environment. By pooling their experiences, each agent can learn from a broader set of transitions than it would gather on its own. Moreover, allowing agents to run asynchronously eliminates the need for strict synchronization, thereby leveraging parallel computational resources more effectively. We further hypothesize that aggregating experiences across agents can reduce training variance across random seeds by smoothing idiosyncratic exploration trajectories, yielding more stable learning dynamics.

In the following, we outline related work in off-policy RL, multi-agent systems, and distributed learning. We then formalize the theoretical foundations of replay-based updates and discuss how the multi-agent approach can potentially mitigate some of the convergence challenges in single-agent scenarios. Subsequently, we present our proposed framework in detail, explaining the architectural decisions, algorithmic implementations, and design trade-offs. Finally, we discuss experimental results in Section 6 and conclude in Section 7.

2. Related Work

Reinforcement learning (RL) has undergone a surge of interest in recent years, culminating in performance breakthroughs in areas ranging from game-playing [1, 5] to robotic control [6, 2] and beyond. This section provides a more comprehensive overview of the literature most relevant to our work, structured around the core themes of off-policy learning, experience replay, multi-agent RL, and distributed or parallelized RL.

2.1 Off-Policy Reinforcement Learning

Off-policy RL methods enable an agent to learn about an optimal policy while following a different behavior policy. Classic Q-learning [7] embodies an off-policy approach by iteratively updating an estimate of the state-action value function Q(s, a) toward the Bellman optimality target. Modern deep RL methods, beginning with DQN [1], extended Q-learning to use neural networks for function approximation, making it feasible to tackle high-dimensional tasks such as playing Atari games from pixel inputs.

Further improvements upon DQN include Double DQN [8], which addresses the overestimation bias inherent in Q learning by using separate networks for action selection and target evaluation. Similarly, Rainbow DQN [9] consolidates multiple enhancements (e.g., prioritized replay, multi-step returns, distributional value functions) into one integrated framework, underscoring the iterative nature of off-policy algorithm refinement. Off-policy principles also appear in actor-critic approaches, such as DDPG [6] and Soft Actor-Critic (SAC) [3], which maintain a replay buffer and update both a policy (actor) and value function (critic) from stored transitions, enabling stable training in continuous action domains.

2.2 Experience Replay and Its Variants

Origins and Motivation. Experience replay was proposed to reuse past transitions and break the temporal correlations in sequential data [4]. Mnih et al. [1] popularized its usage in deep RL, showing that random mini-batch sampling from a replay buffer markedly stabilizes training. By decoupling data collection from policy updates, replay buffers grant RL algorithms a pseudo-i.i.d. perspective, facilitating gradient-based optimization.

Prioritized Replay. A notable extension is prioritized experience replay [10], where transitions with higher TD error (indicating higher learning potential) are sampled more frequently. This technique addresses the inefficiency of uniform sampling when many transitions are less relevant or have near-zero error.

Distributional and Multi-Step Methods. Subsequent work delved into distributional approaches, such as C51 [11] and QR-DQN [12], which approximate the return distribution rather than a single expected value. These methods still rely on experience replay for effective off-policy learning. Similarly, multi-

step returns [9] combine short-term bootstrap updates with replayed transitions to propagate rewards faster. Despite these enhancements, conventional replay usage commonly involves a single agent generating experiences in a single environment instance.

Challenges. Although replay buffers substantially improve sample efficiency, they also introduce complexities. Large buffers can contain outdated transitions from older policies, complicating convergence. Moreover, the data distribution remains limited by whichever single policy is generating it. These challenges motivate the broader coverage approach. Our work aims to diversify replay content by pooling experiences from multiple agents, each potentially exploring different parts of the state-action space.

2.3 Multi-Agent Reinforcement Learning (MARL)

Multi-agent RL (MARL) extends single-agent paradigms to scenarios with multiple agents that may cooperate, compete, or operate entirely independently [13, 14]. In cooperative MARL, agents share a common goal or reward signal [13], whereas competitive or mixed environments involve adversarial or partially aligned objectives [14]. Algorithms like MADDPG [14] use a centralized critic that conditions on the states and actions of all agents, but typically assumes a shared environment.

Research also explores the role of communication among agents [13], joint policy learning [15], and specialized algorithms for partial observability or stable coordination. While many multi-agent frameworks concentrate on how agents interact within a single environment or how they coordinate policies, fewer works examine multiple environment instances per agent. Our approach, by contrast, focuses on asynchronous exploration in parallel environment replicas, united by a shared replay buffer rather than direct inter-agent communication.

2.4 Distributed and Parallelized RL

Scaling Single-Agent Policies. Distributed RL aims to speed up data collection and computation by employing multiple actors or learners that share parameters or periodically synchronize [16, 17, 18]. For instance, in Ape-X [17], many actors feed experience into a central replay buffer, from which a single learner updates the policy. R2D2 [18] extends this to recurrent architectures. These frameworks demonstrate that parallel data gathering and large-scale replay can yield state-of-the-art results. However, they typically revolve around one global policy that is broadcast to all actors.

Independent Agents. In contrast, an independent multi-agent system might train individual policies per agent, each specialized for its own perspective or objective. The synergy lies in sharing experiences rather than a single parameter set. This reduces synchronization overhead while still benefiting from parallel exploration. Our method falls into this category: we allow each agent to maintain separate neural networks, hyperparameters, and exploration schedules. By depositing transitions into a common replay buffer, we unify the data stream while preserving agent autonomy.

Relevance to Our Work. Although many distributed RL techniques emphasize throughput and scaling, our work highlights the benefit of diverse exploration from multiple independent agents. We hypothesize that such diversity can accelerate convergence beyond what a single policy could achieve, even if that policy had the same effective throughput. Empirical evidence supporting this hypothesis appears in multi-robot exploration, multi-player game learning, and distributed policy optimization [5], but systematic study of asynchronous, shared-replay multi-agent systems remains more limited in the literature.

2.5 Summary and Key Contributions

In sum, a large body of research has demonstrated:

- **Off-policy methods** are powerful for reusing past data, but typically hinge on one agent's policy distribution.

- **Experience replay** greatly stabilizes deep RL and can be enhanced with priority sampling, distributional learning, and multi-step returns.
- **Multi-agent RL** often addresses multi-entity environments, focusing on either coordinated or adversarial interactions.
- **Distributed RL** typically uses parallel actors feeding a single learner's replay, or a single policy broadcast to all workers.

By bridging these strands, our work argues for multiple fully independent agents, each interacting with its own environment instance, sharing a replay buffer to cross-pollinate experiences. This perspective aims to combine the efficiency gains of distributed RL with the exploration diversity that arises naturally from multi-agent settings. As we detail next, our approach requires minimal modifications to existing off-policy algorithms, offering a general blueprint for scaling replay-based methods across many parallel agents.

3. Theoretical Foundations

This section formalizes the underpinnings of off-policy learning and explains how multiple agents storing data into a single replay buffer can influence learning dynamics. We begin with standard RL definitions, then move on to off-policy update equations, followed by an analysis of how a shared replay modifies the sampling distribution.

3.1 Preliminaries

We model the environment as a Markov Decision Process (MDP), defined by the tuple $(\mathcal{S}, \mathcal{A}, P, r, \gamma)$. Here, $\mathcal{S}$ denotes the set of possible states, $\mathcal{A}$ denotes the set of actions (discrete or continuous), $P(s'|s,a)$ captures the environment's transition dynamics, $r(s,a)$ is the reward function, and $\gamma \in [0,1)$ is the discount factor. At time t, the agent observes a state $s_t \in \mathcal{S}$, selects an action $a_t \in \mathcal{A}$ and receives a reward $r_t = r(s_t, a_t)$ before transitioning to the next state s_{t+1}. The agent's objective is to maximize the discounted return:

$$G_t \;=\; \sum_{k=0}^{\infty} \gamma^k \, r_{t+k}.$$

3.2 Off-Policy Learning

Off-policy algorithms (e.g., Q-learning) seek to learn an optimal policy π^* for action selection, yet they can gather data from a different (often more exploratory) behavior policy μ. Let $Q_\theta(s,a)$ be a parametric approximation of the state-action value function. The classic Q-learning update is

$$\theta \leftarrow \theta \; - \; \alpha \nabla_\theta \Big(Q_\theta(s,a) - y \Big)^2,$$

where

$$y \; = \; r \; + \; \gamma \max_{a'} Q_{\theta^-}(s',a'),$$

and θ^- are the parameters of a slowly updated target network [1].

3.3 Experience Replay

Experience replay [4] addresses the correlation among sequential samples by storing transitions (s,a,r,s') in a buffer $\mathcal{D}$. By randomly sampling mini-batches from $\mathcal{D}$, the agent approximates an $i.i.d.$ assumption critical for stable gradient-based optimization. This technique also reuses past data, improving sample efficiency.

3.4 Multiple Agents Sharing a Replay Buffer

Now consider N agents, each with a distinct policy π_i and environment instance $\mathcal{E}_i$. Every time agent i takes an action $a_t^{(i)}$ in state $s_t^{(i)}$, we store the resulting transition $(s_t^{(i)}, a_t^{(i)}, r_t^{(i)}, s_{t+1}^{(i)})$ in a shared buffer D. Consequently, the data distribution in D is effectively a mixture of state-action visitation distributions:

$$d_{\text{mix}}(s,a) \; = \; \frac{1}{N} \sum_{i=1}^{N} d^{\pi_i}(s,a).$$

In practice, if agents contribute transitions at different rates (e.g., due to asynchronous execution speed differences), the empirical mixture is closer to a contribution-weighted average.

Each agent A_i then samples mini-batches from $\mathcal{D}$ for its own off-policy updates.

While single-agent convergence proofs typically assume that samples originate from a single evolving behavior policy, the multi-agent scenario introduces an additional layer of complexity due to multiple policies shaping the replay distribution. Nevertheless, empirical evidence suggests that mixing different exploration strategies can lead to faster coverage of $\mathcal{S} \times \mathcal{A}$ often resulting in improved learning speed.

Variance Reduction Intuition. Let $g(s,a)$ denote a stochastic update signal (e.g., a TD error or gradient component) computed from a transition (s,a). Under a single agent, g is drawn from that agent's visitation distribution d^{π_i}. Under shared replay, samples are drawn from the mixture $d_{\text{mix}}(s,a) = \frac{1}{N} \sum_{i=1}^{N} d^{\pi_i}(s,a)$. By the law of total variance,

$$\text{Var}_{d_{\text{mix}}}(g) = \mathbb{E}_i[\text{Var}_{d^{\pi_i}}(g)] + \text{Var}_i(\mathbb{E}_{d^{\pi_i}}[g]) \, .$$

When agents explore different regions, the mixture tends to average out idiosyncratic trajectories. In practice this can reduce sensitivity to any single seed's exploration noise. Moreover, updates average minibatches sampled from , so the effective variance of gradient estimates decreases with both batch size and the diversity of samples in the buffer.

3.5 Implications of Learning from Another Agent's Samples

In a multi-agent off-policy setting, each agent A_i updates its policy π_i using transitions that may originate from a different agent $A_j (j \neq i)$. Formally, if $d^{\pi_j}(s, a)$ denotes the state-action visitation distribution under π_j, then storing (s, a, r, s') in a shared buffer introduces a distribution mismatch when A_i samples from

$$d_{\text{mix}}(s, a) \;=\; \frac{1}{N} \sum_{j=1}^{N} d^{\pi_j}(s, a).$$

Since d^{π_j} might not match d^{π_i}, the transition distribution observed by A_i can deviate from the one it would generate itself. This scenario is inherently off-policy: the behavior policy (one or more $\{\pi_j\}$) differs from the target policy π_i for each agent.

Theoretical analyses of off-policy learning (e.g., Q-learning [7]) generally assume the samples come from a single behavior policy. In multi-agent systems, however, each agent might follow its own exploration scheme, creating a mixture distribution that could, in principle, accelerate coverage of the state-action space. On the other hand, the non-stationarity introduced by multiple, concurrently changing policies complicates convergence proofs. Existing single-agent off-policy results rely on a slowly evolving behavior policy or diminishing exploration schedules [19, 20]. In the multi-agent case, each agent's dynamics may not be stationary if other agents' policies are continuously being updated.

Despite these theoretical challenges, empirical evidence often shows that sharing experience across agents boosts sample efficiency. Intuitively, A_i can quickly learn from beneficial trajectories discovered by A_j without having to find them via its own exploration. The trade-off is that a single agent's learned Q-values or policy gradients reflect a data distribution partly determined by other agents' actions. Future research on multi-agent off-policy convergence seeks to refine conditions under which mixture distributions remain sufficiently rich, yet stable, to ensure reliable learning progress for each agent.

4. Proposed Approach

Building on the theoretical underpinnings discussed in Section 3, we propose an architecture that organizes multiple off-policy agents into a single, shared replay system. Below, we detail the primary components of this framework.

4.1 Algorithm Selection for Discrete and Continuous Actions

While our framework is designed to be agnostic to any off-policy algorithm that employs experience replay, we focus on two representative methods:

- **Double Deep Q-Network (DDQN)** [8], suitable for discrete action spaces. DDQN mitigates overestimation bias by using separate networks for action selection and target Q-value computation.
- **Deep Deterministic Policy Gradient (DDPG)** [6], designed for continuous action spaces. DDPG employs an actor-critic setup, learning a deterministic policy (actor) in conjunction with a Q-value function (critic).

Adopting both DDQN and DDPG highlights the versatility of our asynchronous shared-replay framework, covering a broad range of action-space modalities commonly found in RL tasks.

4.2 Benchmark Environments

To validate the efficacy of our approach, we employ a selection of well-known gym environments that span both discrete and continuous actions:

- **CartPole-v1**: A classic control problem where the agent must balance a pole by moving a cart left or right (discrete actions).
- **Acrobot-v1**: A two-link robotic arm simulation in which the agent attempts to swing the lower link above a target height (discrete actions).
- **MountainCar-v0**: An underpowered car must be driven up a steep hill by building momentum (discrete actions).
- **FrozenLake-v1** (is slippery=True): A stochastic gridworld navigation task with sparse rewards (discrete actions).
- **Taxi-v3**: A gridworld pickup-and-dropoff task with discrete actions and sparse rewards
- **MountainCarContinuous-v0**: A continuous-action variant of MountainCar, demanding careful throttle control to reach the top of the hill (continuous actions).
- **Pendulum-v1**: A continuous-control swing-up task with dense rewards (continuous actions).

These environments collectively provide varying levels of difficulty, state dimensionalities, and reward structures, ensuring a robust evaluation of both DDQN and DDPG within our asynchronous paradigm.

4.3 Comparisons with Vanilla Single-Agent Training

Although the primary aim is to showcase the benefits of multi-agent, asynchronous exploration, we also compare our framework's performance to vanilla single-agent baselines for DDQN and DDPG. Specifically:

- For **DDQN**, the baseline uses a single agent interacting with one environment instance and storing experiences in a private replay buffer.
- For **DDPG**, we similarly train a single agent in isolation for continuous-action tasks.

By contrasting the asynchronous multi-agent results with the standard single-agent versions of these algorithms, we can quantify the impact of parallel exploration and shared replay on convergence speed and sample efficiency. Empirical results in Section 6 confirm that pooling experiences from multiple agents generally leads to faster convergence and reduced training variance relative to traditional single-agent methods, although time-to-threshold improvements are not universal across all environments.

4.4 Multiple Asynchronous Agents

We initialize N agents, each with:

- **Local policy parameters** θ_i , which can follow a value-based (e.g., DQN-type) or actor-critic (e.g., DDPG-type) approach.
- **Independent environment instance** $\mathcal{E}_i$ that is behaviorally identical to other instances but runs asynchronously.
- **Exploration mechanism** such as ϵ-greedy for discrete actions or added noise for continuous controls.

Each agent steps through its environment, collects transitions, and updates its parameters off-policy from samples drawn from the global replay buffer (described next).

4.5 Shared Global Replay Buffer

All agents store transitions in a single buffer $\mathcal{D}$ with capacity $|\mathcal{D}|_{\max}$. When this capacity is reached, older experiences are ejected (e.g., FIFO). This scheme ensures that:

1. Diverse experiences are likely to be maintained, as different agents explore the environment differently.
2. Each agent can rapidly see transitions generated by other agents, augmenting its own exploration path.
3. The overall speed of data accumulation in $\mathcal{D}$ increases approximately linearly with N (assuming sufficient hardware resources for parallelization).

4.6 Parallel Updates

In each environment, agent A_i **:**

1. Observes $s_t^{(i)}$ and selects action $a_t^{(i)}$.
2. Receives reward $r_t^{(i)}$ and transitions to $s_{t+1}^{(i)}$.
3. Appends $(s_t^{(i)}, a_t^{(i)}, r_t^{(i)}, s_{t+1}^{(i)})$ to $\mathcal{D}$.
4. Samples a mini-batch from $\mathcal{D}$ (if sufficiently large) to perform an off-policy update on θ_i.

The agents proceed asynchronously, meaning no agent needs to wait for updates from another. With proper hardware, this setup efficiently utilizes modern multi-core or distributed systems

4.7 Algorithmic Sketch

A high-level pseudocode outlining our proposed approach is provided below. Each of the N agents repeats the same loop, independently storing data and sampling from $\mathcal{D}$ at its own pace.

Algorithm: *Multi-Agent Off-Policy Training with Shared Replay*

1. Initialize global replay buffer $\mathcal{D}$ with capacity $|\mathcal{D}|_{\max}$
2. **for** i = 1 to N **do**
3. Initialize agent A_i parameters θ_i and local environment $\mathcal{E}_i$
4. **end for**

5. **while** not converged **do**
6. **for** each agent A_i **in parallel do**
7. Observe state $s_t^{(i)}$ from $\mathcal{E}_i$
8. Select action $a_t^{(i)}$ using local policy $\pi_i(\cdot;\theta_i)$
9. Execute $a_t^{(i)}$ in $\mathcal{E}_i$, observe reward $r_t^{(i)}$, next state $s_{t+1}^{(i)}$
10. Store $(s_t^{(i)}, a_t^{(i)}, r_t^{(i)}, s_{t+1}^{(i)})$ in $\mathcal{D}$
11. **if** $|\mathcal{D}| \geq$ batch_size **then**
12. Sample mini-batch from $\mathcal{D}$
13. Update θ_i via an off-policy gradient rule (e.g., Q-learning, actor-critic)
14. **end if**
15. **end for**
16. **end while**

Overall, this shared replay scheme seeks to maximize sample diversity and reduce training time, rendering it broadly suitable for a variety of off-policy algorithms and environment configurations.

5. Experimental Setup and Hyperparameters

We keep algorithm hyperparameters (e.g., learning rate, batch size, discount factor, target update schedule) identical across single-agent and shared-replay settings to ensure a fair comparison. The primary difference between settings is the number of agents, with N = 1 for the baseline and N = 4 for shared replay. Unless otherwise specified, the same configuration is used across all environments. We run 5 random seeds per environment and report mean ± standard deviation for all metrics. Learning curves show the seed-averaged return with a 95% confidence interval (CI) band, and returns are plotted with a 10-episode smoothing window. Each episode is capped at 1000 steps for all environments. We train for 1000 episodes in all environments except MountainCarContinuous-v0, which is trained for 200 episodes due to rapid performance saturation.

Table 1: Hyperparameters for DDQN and DDPG Experiments.
(For single-agent baselines, N = 1. For shared-replay runs, N = 4.)

Hyperparameter	**DDQN**	**DDPG**
Learning rate	1×10^{-3}	Actor: 1×10^{-3} / Critic: 1×10^{-3}
Discount factor (γ)	0.99	0.99
Replay buffer size	1×10^{7}	1×10^{7}
Batch Size	128	128
Target network update	Every 100 steps	Soft update (τ = 0.01)
Optimizer	Adam	Adam
Exploration strategy/noise	ϵ-greedy	Ornstein-Uhlenbeck
ϵ start/end/decay	1.0 / 0.01 / 0.9999	—
OU params (θ, σ)	—	0.15, 0.2
Number of agents (N)	4 (shared-replay)	4 (shared-replay)
Environment instances	One per agent	One per agent

5.1 Environments

We evaluate discrete-action DDQN on: CartPole-v1, Acrobot-v1, MountainCar-v0, FrozenLake-v1 (is slippery=True), and Taxi-v3. We evaluate continuous-action DDPG on: MountainCarContinuous-v0 and Pendulum-v1. For shared-replay runs, we use 4 agents and 4 parallel environment instances.

5.2 Metrics

We report:

- **TimeToThreshold**: first episode (using a 100-episode moving average) that reaches a task-specific threshold reward. It is the smallest episode index t such that the 100-episode moving average $\bar{R}_t = \frac{1}{100}\sum_{e=t-99}^{t} R_e$ first meets or exceeds a task-specific threshold.
- **FinalReturn_Last100**: mean return over the last 100 episodes. It is computed as $\frac{1}{100}\sum_{e=T-99}^{T} R_e$, where R_e is the episodic return and T is the final episode
- **AUC Return**: area under the learning curve across all episodes. It is computed by trapezoidal integration over the episode index, $\sum_{e=1}^{T-1}\frac{R_e + R_{e+1}}{2}$, so higher AUC reflects both faster learning and higher returns throughout training.

6. Results and Discussion

This section summarizes the performance of single-agent baselines against the shared-replay multi-agent setting. Results are averaged over 5 seeds; shaded regions in figures denote 95% CI across seeds.

Stability: Across environments, the shared-replay curves often exhibit tighter confidence bands, indicating reduced variance across seeds and more consistent learning dynamics.

6.1 Acrobot-v1 Environment

Figure 1a shows a rapid initial improvement for both settings, with shared replay climbing faster and stabilizing at a higher (less negative) return. The confidence band for shared replay narrows earlier, indicating more consistent learning across seeds.

6.2 CartPole-v1 Environment

Figure 1b shows shared replay accelerating early learning, reaching high returns sooner than the single-agent baseline. After convergence, both exhibit fluctuations, but shared replay maintains a steadier trajectory and tighter uncertainty band. Note that returns can exceed 500 because episodes are capped at 1000 steps.

6.3 MountainCar-v0 Environment

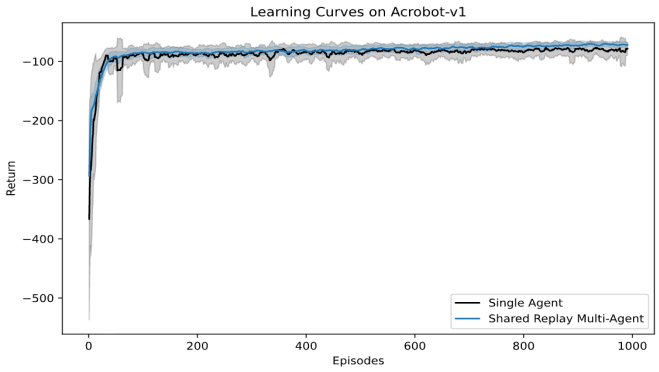
Figure 1c shows large variance early in training, followed by convergence to similar final performance. The single-agent curve improves faster in the first few hundred episodes, while shared replay catches up and remains comparable once both stabilize near the threshold.

6.4 FrozenLake-v1 Environment

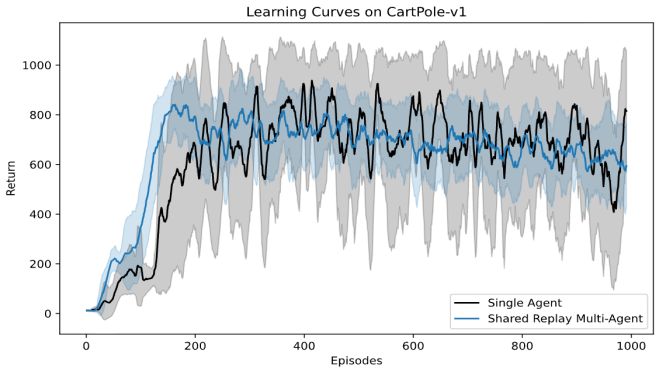
Figure 1d shows that shared replay achieves higher returns sooner and maintains a consistently higher plateau. The uncertainty band is narrower for shared replay, indicating reduced variance across seeds as training progresses in this stochastic environment.

6.5 Taxi-v3 Environment

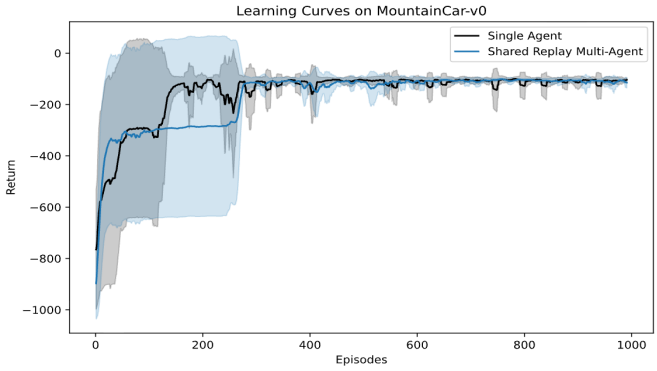
Figure 1e shows a large early gap in favor of shared replay: the shared-replay curve climbs rapidly out of the highly negative reward regime and approaches a near-zero plateau, while the single-agent baseline improves slowly with wide variance. The shared-replay band tightens as training progresses, suggesting more consistent performance across seeds in this sparse-reward task.



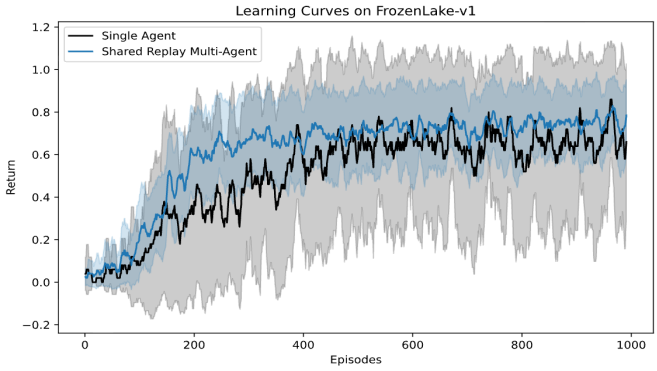
(a) Acrobot-v1



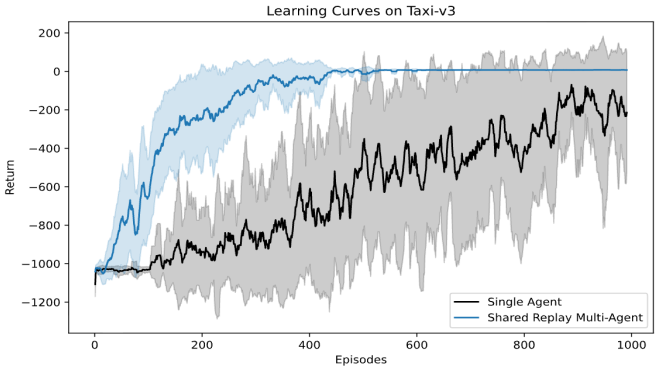
(b) CartPole-v1



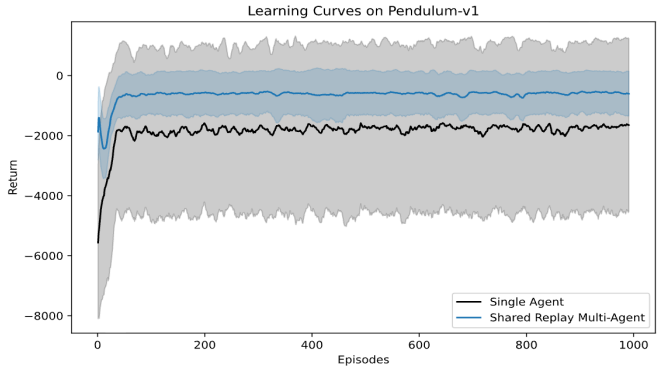
(c) MountainCar-v0



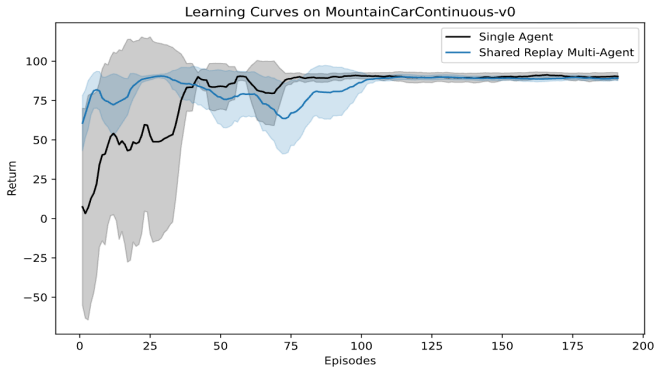
(d) FrozenLake-v1



(e) Taxi-v3



(f) Pendulum-v1



(g) MountainCarContinuous-v0

Figure 1: Learning curves (mean across 5 seeds with 95% CI).

6.6 Pendulum-v1 Environment

Figure 1f shows that shared replay yields a much higher return (less negative) and stabilizes earlier than the single-agent baseline. The shared-replay curve quickly improves from very low returns, then settles into a higher plateau with a narrower confidence band, indicating more consistent performance across seeds in this continuous-control task.

6.7 MountainCarContinuous-v0 Environment

Figure 1g shows that both single-agent and shared-replay DDPG quickly reach the target return of 85 and stabilize near ~ 90 within 200 episodes. The single-agent curve hits the threshold slightly earlier, while shared replay catches up and maintains a similar final plateau with marginally higher AUC, indicating comparable asymptotic performance with a modest advantage in early trajectory integration. Variance is larger in the first few dozen episodes, reflecting exploration noise, but the confidence bands tighten as both policies converge. Because performance plateaus rapidly, we report MountainCarContinuous-v0 based on a 200-episode run without sacrificing interpretability of the learning dynamics.

6.8 General Observations Across Environments

Across the reported gym environments, shared replay reduces variance across seeds and improves AUC in most tasks. TimeToThreshold improvements are not universal: shared replay improves time-to-threshold in 4 of the 7 environments reported here (Acrobot-v1, CartPole-v1, Taxi-v3, and Pendulum-v1), while MountainCar-v0, FrozenLake-v1, and MountainCarContinuousv0 show comparable or slower threshold attainment under our criterion. FinalReturn_Last100 is generally favorable to shared replay, with the largest gains appearing in sparse-reward settings such as Taxi-v3.

Table 2: Summary across environments (mean ± std over 5 seeds).

Environment	Setting	FinalReturn_Last100	AUC_Return	TimeToThreshold	Threshold
Acrobot-v1	Single-agent	-80.308 ± 7.231	-88804.203 ± 3543.789	121.400 ± 13.576	-100
Acrobot-v1	Shared-replay	-71.669 ± 1.532	-83064.172 ± 806.004	113.400 ± 3.362	-100
CartPole-v1	Single-agent	645.238 ± 115.225	617226.500 ± 51643.668	223.200 ± 48.936	475
CartPole-v1	Shared-replay	626.107 ± 45.237	639733.375 ± 40853.125	161.400 ± 19.578	475
MountainCar-v0	Single-agent	-108.734 ± 11.016	-153791.094 ± 72932.844	214.000 ± 69.162	-110
MountainCar-v0	Shared-replay	-110.571 ± 16.409	-171573.406 ± 117617.273	233.750 ± 21.531	-110
FrozenLake-v1	Single-agent	0.680 ± 0.155	506.800 ± 159.086	423.750 ± 128.718	0.7
FrozenLake-v1	Shared-replay	0.754 ± 0.014	612.425 ± 32.517	434.800 ± 198.765	0.7
Taxi-v3	Single-agent	-182.834 ± 170.444	-614660.188 ± 117744.750	NA	7
Taxi-v3	Shared-replay	7.530 ± 0.106	-140081.906 ± 33756.641	617.000 ± 103.322	7
Pendulum-v1	Single-agent	-1700.818 ± 3263.598	-1860486.625 ± 3169615.750	615.333 ± 87.306	-200
Pendulum-v1	Shared-replay	-565.209 ± 806.538	-644154.812 ± 828138.062	439.250 ± 204.559	-200
MountainCarContinuous-v0	Single-agent	90.137 ± 0.740	15753.784 ± 2067.707	123.000 ± 26.823	85
MountainCarContinuous-v0	Shared-replay	89.124 ± 0.688	16584.543 ± 702.945	149.600 ± 38.397	85

7. Conclusion and Future Work

This paper presented an asynchronous multi-agent deep reinforcement learning (RL) framework based on shared experience replay. Multiple agents interact with independent environment instances and contribute transitions to a centralized replay buffer, increasing trajectory diversity and improving learning signals. Across several benchmark environments, the approach yields more stable training (lower seed-to-seed variance) and consistent improvements in AUC, while gains in time-to-threshold appear in multiple tasks but are not universal.

7.1 Contributions and Limitations

The proposed framework offers several key advantages:

- Scalable Parallelism: Asynchronous data collection leverages multi-core and distributed resources without synchronization overhead.
- Enhanced Exploration: Differently initialized agents explore complementary regions of the state-action space, improving replay diversity.
- Sample Efficiency: Transitions discovered by any agent can be reused to accelerate learning, particularly in sparse-reward settings.
- Improved Training Stability: Shared replay reduces sensitivity to initialization and stochastic exploration, improving reproducibility.
- Modular Integration: The method is compatible with a range of off-policy algorithms (e.g., DDQN, DDPG) and can be added to existing pipelines.

Despite these advantages, our empirical evaluation is restricted to a limited set of classic OpenAI Gym benchmarks. While these environments provide standardized testbeds, they do not fully capture the complexity of large-scale, high-dimensional, partially observable, or safety-critical real-world reinforcement learning problems. Consequently, the extent to which the observed performance gains generalize to such settings remains an open question and warrants further investigation.

7.2 Future Directions

Future work will focus on:

- **Stabilizing Asynchrony**: Shared buffers mix evolving policies and can introduce non-stationarity; adaptive sampling, agent-aware prioritization, or ensemble methods may mitigate this effect.
- **Broader Benchmarks**: Extending evaluation to more diverse and challenging environments will better characterize robustness and scalability.
- **Hierarchical and Cooperative Variants**: Incorporating hierarchical control or inter-agent communication may improve performance on long-horizon and multi-stage tasks.
- **Real-World Constraints**: Testing under operational limits (latency, limited compute, partial observability) is necessary to validate practical deployment.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness *et al.* "Human-level control through deep reinforcement learning". *Nature*, vol. 518, no. 7540, pp. 529–533, 2015. DOI: 10.1038/nature14236.

[2] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. "Proximal policy optimization algorithms". *arXiv preprint arXiv:1707.06347*, 2017. https://arxiv.org/abs/1707.06347.

[3] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". *In Proc. Int. Conf. Machine Learning (ICML)*, PMLR, pp. 1861–1870, 2018. https://proceedings.mlr.press/v80/haarnoja18b.html.

[4] L.-J. Lin. "Reinforcement learning for robots using neural networks". Ph.D. dissertation, Carnegie Mellon University, Pittsburgh, PA, 1992. https://apps.dtic.mil/sti/citations/ADA261434.

[5] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre *et al.* "Mastering the game of Go with deep neural networks and tree search". *Nature*, vol. 529, no. 7484, pp. 484–489, 2016. DOI: 10.1038/nature16961.

[6] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez *et al.* "Continuous control with deep reinforcement learning". *In Proc. Int. Conf. Learning Representations (ICLR)*, 2016. https://arxiv.org/abs/1509.02971.

[7] C. J. C. H. Watkins and P. Dayan. "Q-learning". *Machine Learning*, vol. 8, no. 3-4, pp. 279–292, 1992. DOI: 10.1007/BF00992698.

[8] H. Van Hasselt, A. Guez, and D. Silver. "Deep reinforcement learning with double Q-learning". *In Proc. AAAI Conf. Artificial Intelligence*, vol. 30, no. 1, 2016. DOI: 10.1609/aaai.v30i1.10295.

[9] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski *et al.* "Rainbow: Combining improvements in deep reinforcement learning". *In Proc. AAAI Conf. Artificial Intelligence*, vol. 32, no. 1, pp. 3215–3222, 2018. DOI: 10.1609/aaai.v32i1.11796.

[10] T. Schaul, J. Quan, I. Antonoglou, and D. Silver. "Prioritized experience replay". *In Proc. Int. Conf. Learning Representations (ICLR)*, 2016. https://arxiv.org/abs/1511.05952.

[11] M. G. Bellemare, W. Dabney, and R. Munos. "A distributional perspective on reinforcement learning". *In Proc. Int. Conf. Machine Learning (ICML)*, PMLR, pp. 449–458, 2017. https://proceedings.mlr.press/v70/bellemare17a.html.

[12] W. Dabney, G. Ostrovski, D. Silver, and R. Munos. "Implicit quantile networks for distributional reinforcement learning". *In Proc. Int. Conf. Machine Learning (ICML)*, PMLR, pp. 1104–1113, 2018. https://proceedings.mlr.press/v80/dabney18a.html.

[13] J. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson. "Learning to communicate with deep multi-agent reinforcement learning". *In Advances in Neural Information Processing Systems*, vol. 29, pp. 2137–2145, 2016. https://papers.nips.cc/paper/6098-learning-to-communicate-with-deep-multi-agent-reinforcement-learning.

[14] R. Lowe, Y. I. Wu, A. Tamar, J. Harb, P. Abbeel *et al.* "Multi-agent actor-critic for mixed cooperative-competitive environments". *In Advances in Neural Information Processing Systems*, vol. 30, pp. 6379–6390, 2017. https://papers.nips.cc/paper/7217-multi-agent-actor-critic-for-mixed-cooperative-competitive-environments.

[15] J. K. Gupta, M. Egorov, and M. Kochenderfer. "Cooperative multi-agent control using deep reinforcement learning". *In Autonomous Agents and Multiagent Systems*, Springer, pp. 66–83, 2017. DOI: 10.1007/978-3-319-71682-4_5.

[16] V. Mnih, A. P. Badia, M. Mirza, M. Mirza, A. Graves *et al.* "Asynchronous methods for deep reinforcement learning". *In Proc. Int. Conf. Machine Learning (ICML)*, PMLR, pp. 1928–1937, 2016. https://proceedings.mlr.press/v48/mniha16.html.

[17] D. Horgan, J. Quan, D. Budden, G. Barth-Maron, M. Hessel *et al.* "Distributed prioritized experience replay". *In Proc. Int. Conf. Learning Representations (ICLR)*, 2018. https://openreview.net/forum?id=H1Dy---0Z.

[18] S. Kapturowski, G. Ostrovski, J. Quan, M. Hessel, R. Munos *et al.* "Recurrent experience replay in distributed reinforcement learning". *In Proc. Int. Conf. Learning Representations (ICLR)*, 2019. https://openreview.net/forum?id=r1lyTjAqYX.

[19] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*, 2nd ed. Cambridge, MA: MIT Press, 2018. DOI: 10.5555/3312046.

[20] C. Szepesvári. *Algorithms for Reinforcement Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2010. DOI: 10.2200/S00268ED1V01Y201005AIM009.