Evolutionary Population-Based Policy Optimization via High-Throughput Parallel Simulation

Charlotte Ka Yee Yan
Department of Computer Science
Stanford University
ckyy@stanford.edu

Asanshay Gupta
Department of Computer Science
Stanford University
asanshay@stanford.edu

Abstract

Proximal Policy Optimization (PPO) faces significant challenges in multi-agent environments, including hyperparameter sensitivity, suboptimal GPU utilization, and premature convergence due to limited exploration diversity. We address these limitations by implementing a fully GPU-native Population-Based Training (PBT) algorithm that concurrently trains populations of PPO policies with evolutionary selection and mutation. Our approach leverages Madrona's batch simulation engine to run thousands of environment instances in parallel, embedding both neural network weights and hyperparameters directly within GPU memory to eliminate costly CPU-GPU transfers. Each policy in the population maintains unique weights and hyperparameters, enabling diverse exploration strategies. We employ periodic evolutionary updates where elite policies (top 20%) are preserved while under-performing policies are replaced with mutated copies of successful agents. Experimental evaluation in a collaborative two-agent escape room environment demonstrates that larger populations (16-64 agents) consistently outperform smaller configurations and traditional single-policy PPO, with the most significant performance gains occurring between 4 and 8 agents. Optimal mutation rates of 60-80% balance exploration and exploitation effectively. Our results show that population-based training with adequate scale (16-32 agents optimal) offers substantial improvements over single-policy approaches, opening new possibilities for scalable multi-agent reinforcement learning through evolutionary optimization principles.

1 Introduction

Proximal Policy Optimization (PPO) has emerged as one of the most widely used on-policy reinforcement learning algorithms thanks to its clipped-surrogate objective, strong empirical performance, and ease of implementation [Schulman et al. (2017)]. When deployed in complex multi-agent domains, PPO's performance fluctuates dramatically with small adjustments to its learning rate, clipping ratio, and batch size[Adkins et al. (2025)]. Its two-phase loop of trajectory collection followed by sequential stochastic-gradient updates under-utilizes modern GPU hardware[McInroe and Garcin (2025)]. Because policy entropy is gradually reduced over time, the quality of early exploratory trajectories disproportionately influences final outcomes[Shahid et al. (2024)]. Independent parallel agents also tend to converge on similar strategies, limiting exploration diversity across the population[Doulazmi et al. (2025)].

Population-Based Training (PBT) offers a promising solution to these limitations through evolutionary optimization principles [Jaderberg et al. (2017)]. Under-performing members are periodically replaced by mutated copies of top performers, propagating both network weights and hyperparameters and automating what would otherwise be a manual tuning process [Doulazmi et al. (2025)]. This approach

preserves policy diversity through sub-population schedules and migration, balancing exploration and exploitation and avoiding local-optima traps common in vanilla PPO.

Recent advances in GPU-accelerated simulation provide the computational foundation necessary for large-scale population-based training. Madrona's Entity Component System (ECS) architecture enables batch simulation of thousands of environments simultaneously[Shacklett et al. (2023a)]. By making parallel batch execution a fundamental building block, Madrona runs the entire workload on the GPU, yielding an order-of-magnitude performance gain over CPU-only execution. It also spreads the overhead of scene asset storage, rendering, and synchronization across many concurrent simulation requests."

The convergence of population-based optimization with high-throughput GPU simulation presents an opportunity to address PPO's core limitations in multi-agent settings. However, no prior work has integrated evolutionary training methods with GPU-native batch simulation at the scale of thousands of concurrent environments.

We propose applying Population-Based Training to address PPO's core limitations through a dynamic evolutionary approach, using Madrona's high-throughput batch simulation. Through empirical evaluation in a collaborative two-agent environment, we demonstrate that this population-based approach effectively mitigates the failures that plague single-policy PPO implementations, whilst scaling across available hardware to reduce wall-clock training time without sacrificing exploration diversity. This evolutionary cycle of selection and mutation naturally balances exploration against exploitation, potentially proving that parallel PBT is as viable as, if not more effective than, the current status quo, which is PPO, for complex multi-agent environments. We present our code on GitHub at https://github.com/SuperAce100/madrona_escape_room_pbt

2 Related Work

Population-Based Training (PBT) has emerged as a powerful paradigm for addressing hyperparameter sensitivity in deep reinforcement learning. The original PBT framework introduced online optimization that dynamically evolves both network weights and hyperparameters throughout training[Jaderberg et al. (2017)]. Recent advances have significantly expanded this foundation: Multiple-Frequencies PBT addresses premature convergence issues through variable evolution frequencies[Doulazmi et al. (2025)], while GPU-accelerated population-based RL has demonstrated successful sim-to-real transfer on robotic manipulation tasks[Shahid et al. (2024)]. The broader evolutionary RL field has seen substantial developments, with comprehensive surveys providing systematic frameworks for understanding how evolutionary algorithms enhance RL through hyperparameter optimization and policy diversity maintenance[Li et al. (2024b,a)]. Specialized frameworks like EvoRL now provide GPU-accelerated tools specifically designed for evolutionary reinforcement learning[Liang et al. (2025)].

The development of GPU-accelerated simulation environments has been crucial for making PBT practical at scale. Beyond Madrona's ECT architecture[Shacklett et al. (2023b)], several specialized simulators have achieved dramatic speedups: Pgx for board games achieves 10-100x acceleration over Python implementations[Koyamada et al. (2023)], transportation simulators achieve 88.92x speedup[Zhang et al. (2024)], and JAX-LOB enables large-scale financial applications[Frey et al. (2023)]. The broader landscape of parallel RL training encompasses classical distributed architectures with parallel actors and learners[Nair et al. (2015)] and modern GPU-native implementations such as Parallel Q-Learning that achieve superior wall-clock performance[Li et al. (2023)]. Comprehensive surveys categorize these acceleration methodologies into simulation parallelism, computing parallelism, and distributed synchronization mechanisms[Liu et al. (2024a)]. These advances demonstrate the computational foundation necessary for large-scale population-based approaches.

Multi-agent RL presents unique challenges that population-based approaches are well-suited to address, with recent comprehensive surveys providing theoretical foundations for cooperative and competitive scenarios[Huh and Mohapatra (2023); Zhang et al. (2019)]. MAZero combines centralized modeling with Monte Carlo Tree Search for multi-agent coordination[Liu et al. (2024b)], while applications like distributed satellite routing demonstrate scalability benefits in large-scale network optimization[Lozano-Cuadra and Soret (2024)]. Hyperparameter sensitivity remains a critical challenge, with extensive studies demonstrating that choices dramatically affect performance and advocating for principled optimization approaches over manual tuning[Parker-Holder et al. (2022)].

Recent work has explored evolutionary algorithms for hyperparameter optimization[Tani et al. (2020)] and curriculum learning integration[Zhengfeng et al. (2024)], showing how evolutionary principles can be applied beyond traditional parameter tuning to improve entire training processes.

3 Method

We implement a fully GPU-based PBT algorithm that trains a population of N policies concurrently. Each policy is parameterized by a shared neural architecture but maintains unique weights θ_i and hyperparameters $\lambda_i = \{\alpha_i, c_{e,i}, \gamma_i\}$ (learning rate, entropy coefficient, discount factor). At initialization, we spawn this population with diversified starting conditions: weights are sampled as $\theta_i \sim \mathcal{N}(0,\, \sigma^2 I)$ and hyperparameters are drawn from log-uniform priors to encourage broad exploration across the parameter space. All policies implement the same PPO actor-critic model, but because their hyperparameter values and initializations differ, ensuring no two agents begin identically. This diversification allows the population to explore a broad range of behaviors from the start, mitigating premature convergence to a single strategy.

3.1 Parallel Training and Experience Collection

During training, each policy i is assigned a disjoint shard of M parallel environment instances, enabling concurrent experience accumulation. We employ an asynchronous training loop in which all policies collect experience in parallel and periodically synchronize for evaluation and evolution. Each agent interacts with its own batch of environments, collecting experience trajectories independently of the others.

In each iteration of the loop, every policy runs a PPO update on the latest batch of trajectory data from its assigned environments, collected in parallel on GPU. This design allows all agents to train concurrently, fully utilizing the parallelism available. Each policy improves on its own experience shard, while the diversity of initial conditions and interactions across the population prevents convergence to suboptimal local strategies.

3.2 Batch Simulation with Madrona

Our framework leverages the Madrona batch simulation engine to simulate $N \times M$ environment instances on GPU all in parallel. At each simulation tick, observations are exported as GPU-resident tensors $\mathbf{O}_i \in \mathbb{R}^{M \times d_o}$ for each policy i, which feed directly into the policy's neural network to compute action tensors $\mathbf{A}_i \in \mathbb{R}^{M \times d_a}$. These actions are then fed back into the simulator for the next step, creating a tight integration that obviates costly CPU-GPU transfers.

A forward pass is executed for each policy's neural network on this collected batch of observations, and the resulting actions are inserted back into Madrona's action tensor buffers. Because the data never leaves GPU memory, the overhead between simulation and model inference is minimal.

3.3 GPU-Native Updates

We embed both neural weights θ_i and hyperparameters λ_i within Madrona as GPU components. Each policy in the population is associated with its subset of environment instances through components that represent policy-specific data. In practice, we bind each policy's neural network weights and training hyperparameters into the simulation as GPU-resident tensors attached to the entities and world contexts they control.

This design allows PBT operations like replacing a policy's weights or altering its hyperparameters to be carried out on the GPU. During evolution, policies are replaced as follows:

$$\theta_j \leftarrow \theta_k, \quad \lambda_j \leftarrow \lambda_k + \delta \tag{1}$$

where k is the index of an elite policy. By executing replacement in place, we maintain continuous simulation across all $N \times M$ environments and avoid environment reinitialization or CPU intervention.

3.4 Evolutionary Selection and Mutation Strategy

We evaluate policy performance at regular intervals to drive the evolutionary aspect of PBT. At every fixed interval of T training iterations (typically every 500 PPO updates), we compute a fitness score

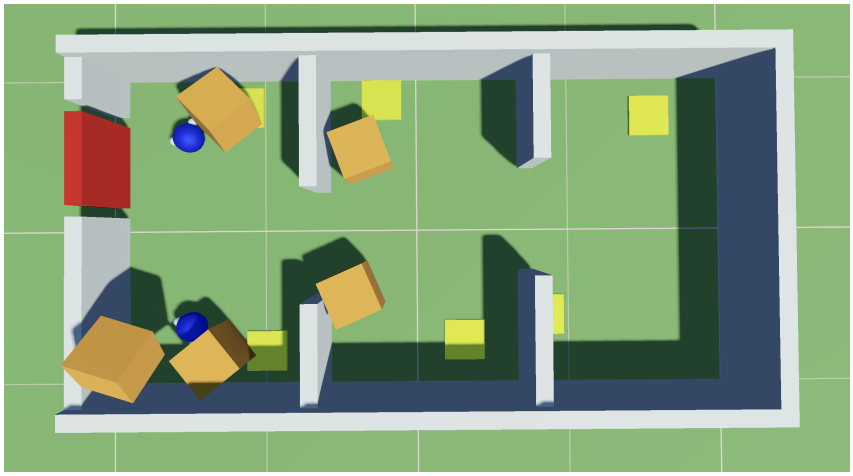

Figure 1: Madrona Escape Room Environment

f_i for each policy based on cumulative reward obtained over the last T minibatches. This periodic evaluation yields a scalar performance score used to rank the population.

We then perform a survival of the fittest update: we rank all policies and select the top $p_e \cdot N$ (roughly 20%) as *elites* to be retained without modification.

For each policy in the rest of the group, we execute an exploit-and-explore operation:

1. **Exploitation**: Copy the neural network parameters from an elite policy: $\theta_j \leftarrow \theta_{\text{elite}}$
2. **Exploration**: Perturb the hyperparameters with small random mutations:

$$\lambda'_j = \lambda_{\text{elite}} \times \exp(\delta), \quad \delta \sim \text{Uniform}(-\epsilon, +\epsilon) \tag{2}$$

where ϵ is a small mutation scale (typically $\pm 10\%$). This ensures the new policy starts from strong network weights but explores a slightly different learning regime than its parent. Algorithm 1 shows the full training loop.

Algorithm 1 Population Based Training Algorithm

Initialize population $\{\theta_i, \lambda_i\}_{i=1}^{N}$
Distribute M simulation instances per policy in Madrona ECS
for iteration = 1 to max_iter **do**
  for $i = 1$ **to** N (in parallel) **do**
    Collect M trajectories via synchronous PPO on θ_i
  end for
  if mod(iteration, T) == 0 **then**
    Compute fitness f_i for each policy
    Select elites and poor performers via ranking
    for each poor policy j **do**
      $\theta_j \leftarrow \theta_k$ from sampled elite k
      $\lambda_j \leftarrow \lambda_k \times \exp(\delta)$, $\delta \sim U(-\epsilon, \epsilon)$
    end for
  end if
end for=0

4 Experimental Setup

4.1 Environment

We evaluate the PBT algorithm in the challenging Madrona Escape Room environment, which requires collaborative strategies from multiple agents to press pressure plates and open gates to

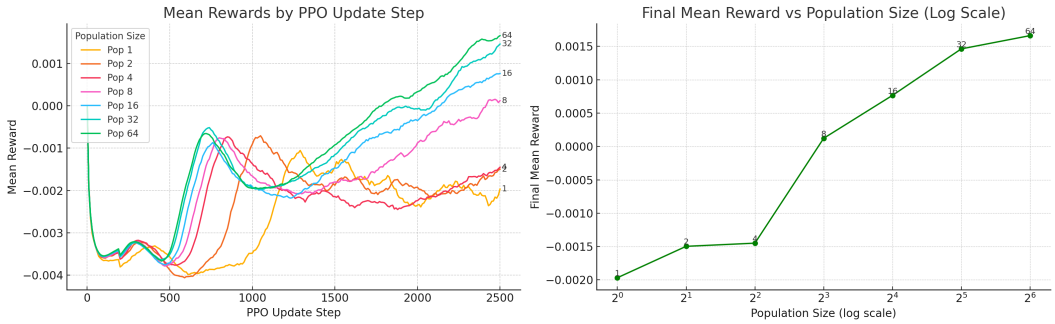

Figure 2: **Left:** Smoothed mean rewards over PPO update steps for different population sizes. **Right:** Final mean reward vs. population size (log scale).

Population Size	Final Mean Reward
1	−0.00197
2	−0.00150
4	−0.00145
8	0.00012
16	0.00077
32	0.00142
64	0.00169

Table 1: Final mean reward achieved after 2500 PPO update steps.

escape Shacklett et al. (2023b). The environment is a 3D world with two agents navigating through a sequence of three rooms arranged along the Y-axis, as can be seen in Figure 1. Agents must coordinate to step on buttons or push blocks onto them to open doors and progress.

At each timestep, agents receive structured observations that include their position, the location of interactive objects (buttons and blocks), local geometry via lidar-like sensing, and binary indicators of environment state (e.g., door open, object held). Actions consist of continuous movement and rotation commands, as well as a discrete grab/release control for interacting with objects. At time (in steps) t, if the maximum distance an agent has traveled is y_{max}, the reward is given by:

$$r_t = 0.05 y_{max} - 0.005t \tag{3}$$

This encourages agents to continually explore deeper into the environment by penalizing them for time taken before completing the level.

5 Results

We investigate the effect of two critical hyperparameters in the PBT algorithm: population size and mutation rate. We report results as the mean episodic reward across PPO update steps, as well as the time to convergence with comparisons.

5.1 Population Size

To assess how the population size influences learning dynamics, we conducted a series of experiments with sizes ranging from 1 to 64. All runs used identical settings for learning rate, entropy coefficient, value loss weight, and number of update steps (2500), which can be seen in Appendix A.Table 1 shows the results of these experiments.

5.1.1 Quantitative Results

Table 1 and Figure 2 (right) quantify how final mean reward scales with population size. As shown, increasing the population from 1 to 64 consistently improves the final reward, from −0.00197 (1

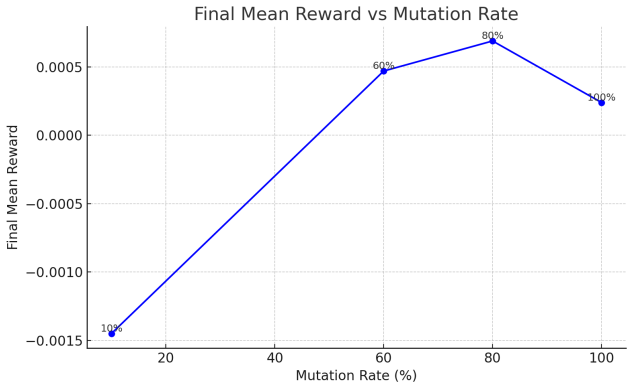

Figure 3: Mean reward over PPO update steps for different mutation rates.

policy) to 0.00169 (64 policies). The most significant jump occurs between population sizes 4 and 8, where performance shifts from negative to positive mean reward. Beyond 16 policies, improvements become more incremental: the 32-policy and 64-policy configurations differ by just 0.00027.

Moreover, plotting these values on a log scaled x-axis reveals an approximately logarithmic trend in reward improvement. This suggests diminishing marginal returns as more agents are added. This has practical implications for resource allocation, as increasing population size incurs computational cost but may offer limited benefit beyond a certain scale.

5.1.2 Qualitative Analysis

The learning curves in Figure 2 (left) further illustrate the dynamics introduced by population scaling. Larger populations not only achieve higher final rewards, but also tend to escape the early plateau phase more quickly. Populations of size 16 and above exhibit smoother, more consistent ascent in reward, likely due to greater policy diversity and exploratory breadth.

Interestingly, while all runs start from similar performance baselines, smaller populations often plateau or oscillate around suboptimal rewards. This may be attributed to insufficient genetic diversity and premature convergence, where poor policies dominate the population before useful traits can propagate. In contrast, larger populations appear more robust to noise and mutation variance, stabilizing around higher-reward regions of the policy space.

However, the marginal improvements from Pop 32 to Pop 64, coupled with increased variance in early training, hint at a tradeoff: overly large populations may introduce noise that slows down convergence or saturates the benefit of diversity. This observation points to the need for smarter selection and adaptation strategies in high-population settings, rather than blind scaling.

5.2 Mutation Rate

We next examine how the mutation rate affects policy evolution and performance, fixing the population size at 4 to isolate the impact of this parameter. We vary the mutation rate across $\{10\%, 60\%, 80\%, 100\%\}$ while keeping other settings constant. As illustrated in Figure 3, performance is highly sensitive to mutation rate.

The lowest mutation rate (10%) results in slower adaptation and lower final reward, likely due to limited exploration and overfitting to suboptimal policies. Conversely, a 100% mutation rate causes instability and erratic learning, as policies are frequently perturbed, limiting convergence. Intermediate rates (60% and 80%) yield the best performance, achieving a balance between exploitation of high-reward policies and continued exploration of the search space.

These findings highlight the need for careful tuning of mutation hyperparameters in PBT. Furthermore, they suggest that adaptive mutation schedules or meta-optimization of mutation intensity may be promising directions for future work.

Mutation Rate (%)	Final Mean Reward
10	−0.00145
60	0.00047
80	0.00069
100	0.00024

Table 2: Final mean reward for different mutation rates at fixed population size (4 policies).

5.3 Convergence Time

To understand how population size influences convergence efficiency, we define convergence as the last PPO update step where the smoothed average reward crosses from negative to non-negative. This definition captures the first point at which we can expect the majority of the rollouts to complete successfully. We observe that larger populations not only achieve higher final rewards but also converge more consistently and earlier in training.

As shown in Table 3, population sizes of 1, 2, and 4 fail to ever sustain positive rewards, whereas populations of 8 or more reliably cross the zero threshold. Notably, populations of 32 and 64 converge earlier than those with fewer agents, indicating that larger populations help overcome initial optimization instability and exploration challenges.

The accompanying plot (Figure 4) visualizes the convergence step for each population size. However, as we see in Table 3, this does not actually correspond to the wall clock time, where all 4 policies are quite similar in their final values.

Population Size	Convergence Step	Wall Time (min)	Final Mean Reward
1	N/A	N/A	−0.00197
2	N/A	N/A	−0.00150
4	N/A	N/A	−0.00145
8	2420	16.63	0.00012
16	2140	23.26	0.00077
32	2090	18.11	0.00146
64	1810	23.02	0.00166

Table 3: Convergence Step vs Population Size

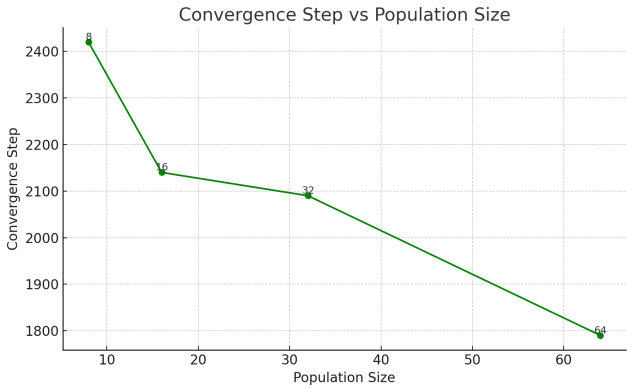

Figure 4: Convergence Step vs Population Size

6 Discussion

Our experimental results demonstrate several key successes in scaling population-based training to multi-agent reinforcement learning. Most significantly, we showed that larger populations consistently

achieve better final performance, with improvements continuing up to 64 agents. The logarithmic scaling relationship we observed (Figure 2) indicates that while gains diminish, they remain substantial even at larger population sizes. The evolutionary selection mechanism proved highly effective at maintaining policy diversity while propagating successful strategies. Populations of 8 or more agents reliably achieved positive rewards and completed the collaborative task, while smaller populations consistently failed to converge. This suggests that the exploration benefits of population diversity are crucial for overcoming the coordination challenges inherent in multi-agent learning. Our mutation rate experiments successfully identified the importance of balanced exploration-exploitation tradeoffs. The 60-80% mutation rate range yielded optimal performance, demonstrating that moderate perturbations allow policies to build upon successful strategies while still exploring new approaches. This finding provides practical guidance for practitioners implementing population-based training. The GPU-native implementation achieved our goal of making population-based training computationally tractable. By embedding both neural network weights and hyperparameters directly within Madrona's ECS architecture, we eliminated costly CPU-GPU transfers and enabled seamless scaling across available hardware. The wall clock times reported in Table 3 show that training 32-64 agents requires comparable time to training smaller populations, making large-scale evolutionary approaches practically viable.

Despite these successes, several constraints emerged. Performance gains diminished beyond 32-64 agents, suggesting fundamental scalability limits due to increased mutation noise and resource contention. Our evaluation was limited to a single environment, raising questions about generalizability to other multi-agent tasks. The sensitivity to mutation rate also highlights a key limitation: the approach still requires manual tuning of mutation parameters, indicating that fully automated hyperparameter optimization remains elusive. Additionally, while larger populations improved training efficiency in terms of iterations, wall clock time benefits were modest, suggesting suboptimal GPU resource utilization at scale.

7 Conclusion

Population-based training offers substantial improvements over traditional single-policy PPO when implemented at sufficient scale (16-32 agents optimal). However, it requires adequate population size to be effective and is not universally superior to traditional methods. The integration with GPU-accelerated simulation opens new possibilities for large-scale multi-agent learning, though future work should focus on adaptive mutation strategies and broader task generalizability.

7.1 Directions for Future Work

Several promising directions remain for exploration:

- **Advanced Selection Mechanisms**: Future iterations could incorporate selection strategies grounded in behavioral diversity metrics, such as trajectory embeddings or state visitation frequency, rather than relying solely on reward-based rankings. This may improve exploration and policy robustness.
- **Adaptive Population Sizing**: Introducing dynamic population management that scales based on convergence speed or diversity collapse could lead to more efficient use of compute and better adaptation across training regimes.
- **Cross-Environment Generalization**: To assess policy robustness and transferability, it is valuable to evaluate evolved policies across a broader suite of multi-agent tasks, particularly those with distinct coordination dynamics or environmental structures.

8 Team Contributions

- **Charlotte:** Charlotte led the design of the evolutionary component of the algorithm and implemented a scratch version with mock policy weights.
- **Asanshay:** Asanshay integrated this algorithm into the final Escape Room Madrona environment and conducted the experiments.

Changes from Proposal: This is similar to the plan we laid out in the proposal, with a few shifts to account for the quirks of integrating with the Madrona simulator.

References

Jacob Adkins, Michael Bowling, and Adam White. 2025. A Method for Evaluating Hyperparameter Sensitivity in Reinforcement Learning. arXiv:2412.07165 [cs.LG] https://arxiv.org/abs/2412.07165

Waël Doulazmi, Auguste Lehuger, Marin Toromanoff, Valentin Charraut, Thibault Buhet, and Fabien Moutarde. 2025. Multiple-Frequencies Population-Based Training. arXiv:2506.03225 [cs.LG] https://arxiv.org/abs/2506.03225

Sascha Frey, Kashif Rasul, Yuriy Nevmyvaka, Jiahao Weng, Christoph Bergmeir, Nicolas Chapados, Yoshua Bengio, and Antoine Gobeil. 2023. JAX-LOB: A GPU-Accelerated limit order book simulator to unlock large scale reinforcement learning for trading. arXiv:2308.13289 [cs.LG] https://arxiv.org/abs/2308.13289

Dom Huh and Prasant Mohapatra. 2023. Multi-agent Reinforcement Learning: A Comprehensive Survey. arXiv:2312.10256 [cs.LG] https://arxiv.org/abs/2312.10256

Max Jaderberg, Valentin Dalibard, Simon Osindero, Wojciech M. Czarnecki, Jeff Donahue, Ali Razavi, Oriol Vinyals, Tim Green, Iain Dunning, Karen Simonyan, Chrisantha Fernando, and Koray Kavukcuoglu. 2017. Population Based Training of Neural Networks. arXiv:1711.09846 [cs.LG] https://arxiv.org/abs/1711.09846

Sotetsu Koyamada, Shinri Okano, Soichiro Nishimori, Yu Murata, Keigo Habara, Haruka Kita, and Shin Ishii. 2023. Pgx: Hardware-Accelerated Parallel Game Simulators for Reinforcement Learning. arXiv:2303.17503 [cs.LG] https://arxiv.org/abs/2303.17503

Pengyi Li, Jianye Hao, Hongyao Tang, Yan Zheng, and Xian Fu. 2024a. Bridging Evolutionary Algorithms and Reinforcement Learning: A Comprehensive Survey on Hybrid Algorithms. arXiv:2401.11963 [cs.LG] https://arxiv.org/abs/2401.11963

Yajing Li, Yaochu Jin, Ran Cheng, and Kaisa Miettinen. 2024b. Reinforcement Learning-assisted Evolutionary Algorithm: A Survey and Research Opportunities. arXiv:2308.13420 [cs.LG] https://arxiv.org/abs/2308.13420

Zechu Li, Tao Chen, Lingfeng Sun, Zhuo Zhang, Weinan Zhang, and Yu Wang. 2023. Parallel Q-Learning: Scaling Off-policy Reinforcement Learning under Massively Parallel Simulation. arXiv:2307.12983 [cs.LG] https://arxiv.org/abs/2307.12983

Zhenyu Liang, Kaiyu Chen, Yifan Zhong, Zhengyue Zhao, Qiwei Ye, Cheng He, Ran Cheng, and Yue Zhang. 2025. EvoRL: A GPU-accelerated Framework for Evolutionary Reinforcement Learning. arXiv:2501.15129 [cs.LG] https://arxiv.org/abs/2501.15129

Qihan Liu, Jianing Ye, Xiaoteng Ma, Jun Yang, Bin Liang, and Chongjie Zhang. 2024b. Efficient Multi-agent Reinforcement Learning by Planning. arXiv:2405.11778 [cs.LG] https://arxiv.org/abs/2405.11778

Zhihong Liu, Xin Xu, Jia Xu, and Kangning Yin. 2024a. Acceleration for Deep Reinforcement Learning using Parallel and Distributed Computing: A Survey. arXiv:2411.05614 [cs.LG] https://arxiv.org/abs/2411.05614

Federico Lozano-Cuadra and Beatriz Soret. 2024. Multi-Agent Deep Reinforcement Learning for Distributed Satellite Routing. arXiv:2402.17666 [cs.LG] https://arxiv.org/abs/2402.17666

Trevor McInroe and Samuel Garcin. 2025. PixelBrax: Learning Continuous Control from Pixels End-to-End on the GPU. arXiv:2502.00021 [cs.LG] https://arxiv.org/abs/2502.00021

Arun Nair, Praveen Srinivasan, Sam Blackwell, Cagdas Alcicek, Rory Fearon, Alessandro De Maria, Vedavyas Panneershelvam, Mustafa Suleyman, Charles Beattie, Stig Petersen, Shane Legg, Volodymyr Mnih, Koray Kavukcuoglu, and David Silver. 2015. Massively Parallel Methods for Deep Reinforcement Learning. arXiv:1507.04296 [cs.LG] https://arxiv.org/abs/1507.04296

Jack Parker-Holder, Raghu Rajan, Xingyou Song, André Biedenkapp, Yingjie Miao, Theresa Eimer, Baohe Zhang, Vu Nguyen, Roberto Calandra, Aleksandra Faust, Frank Hutter, and Marius Lindauer. 2022. Hyperparameters in Reinforcement Learning and How To Tune Them. arXiv:2306.01324 [cs.LG] https://arxiv.org/abs/2306.01324

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal Policy Optimization Algorithms. arXiv:1707.06347 [cs.LG] https://arxiv.org/abs/1707.06347

Brennan Shacklett, Luc Guy Rosenzweig, Zhiqiang Xie, Bidipta Sarkar, Andrew Szot, Erik Wijmans, Vladlen Koltun, Dhruv Batra, and Kayvon Fatahalian. 2023a. An Extensible, Data-Oriented Architecture for High-Performance, Many-World Simulation. *ACM Trans. Graph.* 42, 4, Article 90 (July 2023), 13 pages. https://doi.org/10.1145/3592427

Brennan Shacklett, Luc Guy Rosenzweig, Zhiqiang Xie, Bidipta Sarkar, Andrew Szot, Erik Wijmans, Vladlen Koltun, Dhruv Batra, and Kayvon Fatahalian. 2023b. An Extensible, Data-Oriented Architecture for High-Performance, Many-World Simulation. *ACM Trans. Graph.* 42, 4 (2023).

Asad Ali Shahid, Yashraj Narang, Vincenzo Petrone, Enrico Ferrentino, Ankur Handa, Dieter Fox, Marco Pavone, and Loris Roveda. 2024. Scaling Population-Based Reinforcement Learning with GPU Accelerated Simulation. arXiv:2404.03336 [cs.RO] https://arxiv.org/abs/2404.03336

Laurits Tani, Gert Aarts, Charilaos Akasiadis, and Biagio Lucini. 2020. Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics. arXiv:2011.04434 [cs.LG] https://arxiv.org/abs/2011.04434

Jun Zhang, Jinhu Lü, Jingru Yao, Jintao Ke, and Honggang Wang. 2024. A GPU-accelerated Large-scale Simulator for Transportation System Optimization Benchmarking. arXiv:2406.10661 [cs.LG] https://arxiv.org/abs/2406.10661

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. 2019. Multi-Agent Reinforcement Learning: A Selective Overview of Theories and Algorithms. arXiv:1911.10635 [cs.LG] https://arxiv.org/abs/1911.10635

Zhao Zhengfeng, Bernardo N. Ramos, Muhammad Usama, and Diego Perez-Liebana. 2024. Online Optimization of Curriculum Learning Schedules using Evolutionary Optimization. arXiv:2408.06068 [cs.LG] https://arxiv.org/abs/2408.06068

A Chosen Hyperparameters

Hyperparameter	Value
Experiment Type	`escape_room`
Elite Fraction	0.2
Learning Rate	1×10^{-4}
Discount Factor (γ)	0.998
Entropy Coefficient	0.01
Value Loss Coefficient	0.5
Channels	256
Separate Value Network	False
Steps per Update	40
FP16 Enabled	True

Table 4: Consistent hyperparameters shared across all experiments. These values were fixed to isolate the effects of population size and mutation parameters.