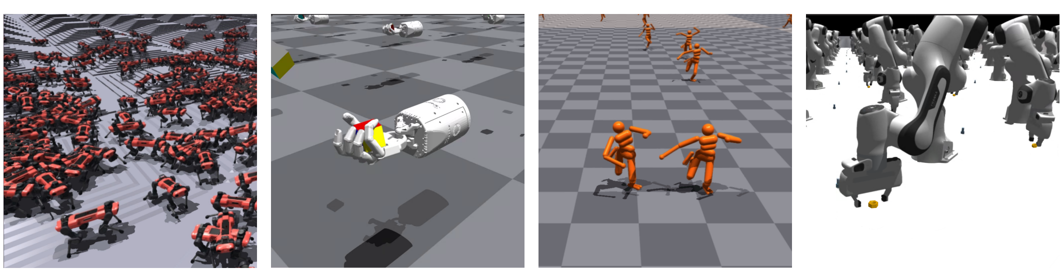
Scaling Population-Based Reinforcement Learning with GPU Accelerated Simulation

Anonymous Author(s)
Affiliation
Address
email

Abstract

In recent years, deep reinforcement learning (RL) has shown its effectiveness in solving complex continuous control tasks like locomotion and dexterous manipulation. However, this comes at the cost of an enormous amount of experience required for training, exacerbated by the sensitivity of learning efficiency and the policy performance to hyperparameter selection, which often requires numerous trials of time-consuming experiments. This work introduces a Population-Based Reinforcement Learning (PBRL) approach that exploits a GPU-accelerated physics simulator to enhance the exploration capabilities of RL by concurrently training multiple policies in parallel. The PBRL framework is applied to three state-of-the-art RL algorithms – PPO, SAC, and DDPG – dynamically adjusting hyperparameters based on the performance of learning agents. The experiments are performed on four challenging tasks in Isaac Gym – *Anymal Terrain*, *Shadow Hand*, *Humanoid*, *Franka Nut Pick* – by analyzing the effect of population size and mutation mechanisms for hyperparameters. The results demonstrate that PBRL agents outperform non-evolutionary baseline agents across tasks essential for humanoid robots, such as bipedal locomotion, manipulation, and grasping in unstructured environments. The trained agents are finally deployed in the real world for the *Franka Nut Pick* manipulation task. To our knowledge, this is the first sim-to-real attempt for successfully deploying PBRL agents on real hardware. Code and videos of the learned policies are available on our project website.



(a) *Anymal Terrain* (b) *Shadow Hand* (c) *Humanoid* (d) *Franka Nut Pick*

Figure 1: Simulated experiments are performed on four Isaac Gym benchmark tasks: (a) *Anymal Terrain*, to teach a quadruped robot to navigate uneven terrain; (b) *Shadow Hand*, which involves manipulating a cube to a desired orientation with a robot hand; (c) *Humanoid*, for bipedal locomotion; and (d) *Franka Nut Pick*, where the goal is to grasp and lift a nut from a random location on a work surface.

1 Introduction

Many domains have seen tremendous advancements of reinforcement learning (RL) applications in recent years, ranging from playing challenging games [31, 5] to learning high-dimensional continuous control in robotics [30, 24, 29]. Tasks such as dexterous manipulation [4], legged locomotion [22], and mobile navigation [16] have been learned using deep RL. A primary challenge in training RL policies is the need for large amounts of training data. RL methods rely on effective exploration to discover control policies, which can be particularly challenging when operating in high-dimensional continuous spaces [35]. Moreover, the performance of the learned policy is highly dependent on the tedious tuning of hyperparameters. Hyperparameter tuning can be a very time-consuming process, often requiring many manual trials to determine the best values for the specific task and the learning environment. One way to deal with the problem of data inefficiency is to train in simulation before transferring to reality [32, 2, 23]. However, the time required to train the policy in simulation increases significantly with the task complexity. For example, in [2], learning a block re-orientation task with a robot hand took around 14 days and enormous computing resources. In addition, policies trained in simulation often fail to perform on a real system due to discrepancies between the simulation and the real world. Recent advances in GPU-accelerated simulation, such as Isaac Gym [21, 11], have made it possible to run thousands of parallel environments on a single GPU, which reduces the training times significantly. However, successfully training RL policies still requires carefully tuned hyperparameters to explore efficiently.

1.1 Related Works

1.1.1 Massively Parallel Simulation

The advent of GPU-based simulation has significantly improved simulation throughput by incorporating massive parallelism on a single GPU [21, 19]. A number of recent works have exploited this parallelism to demonstrate impressive performance on challenging control problems using RL [11, 3, 27]. However, almost all recent works use the same algorithm, i.e. Proximal Policy Optimization (PPO) [28] to train RL policies; other common approaches include off-policy techniques, e.g. Soft Actor-Critic (SAC) [10] and Deep Deterministic Policy Gradient (DDPG) [20]. While simple and effective, all these algorithms require a range of hyperparameters that need to be tuned for each task to ensure sufficient exploration and stabilize training.

1.1.2 Population-Based Reinforcement Learning

Population-based approaches offer a promising solution to deal with exploration and hyperparameter tuning by training a set of policies as opposed to a single policy. Multiple agents can be used to collect diverse experiences that improve robustness and stabilize training by dynamically adapting the hyperparameters. Some prior works have shown remarkable results in employing these approaches to train deep RL policies in domains like strategy games and multi-agent interaction [34, 14, 9]. However, there is almost no existing research investigating PBRL methods for robotics. This is due to the fact that the computational complexity and training time of these approaches increase linearly with the number of agents on CPU-based simulators like MuJoCo [33], requiring multiple worker machines with separate simulation instances to speed up data collection. Isaac Gym allows simulating thousands of robots in parallel, giving access to a vast amount of experience data, rendering it suitable to efficiently train a population of RL agents.

Training various RL agents provides a mechanism for meta-optimization, utilizing the potential of both learning and evolution [1]. One successful example of PBRL methods is population-based training (PBT) [15], which allows training multiple policies concurrently to enhance the exploration capabilities of the agents in generating diverse behaviors. PBT trains a population of agents with different hyperparameters and uses a genetic algorithm to update the population periodically. Recently, DexPBT, a decentralized PBRL approach has been proposed to learn dynamic manipulation between two hand-arm systems using parallel simulations [26]. The authors developed a decentralized implementation to evolve agents in distributed computing environments using on-policy RL, achieving impressive results in dexterous manipulation. However, *sim-to-real* transfer has not been performed, highlighting the complexity of deploying policies on real systems.

In contrast, this work targets a broader range of real-world tasks including locomotion and manipulation, and transfers the policy onto a real robot without any adaptation phase. In addition, the

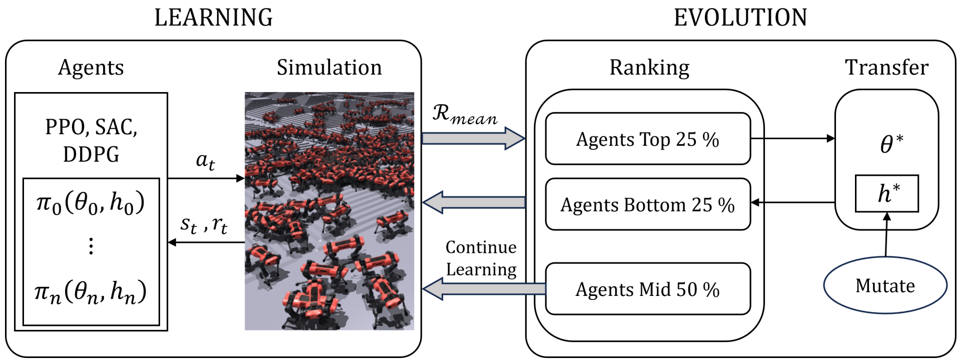


Figure 2: PBRL framework used to learn robotic manipulation tasks through a combination of RL, evolutionary selection, and GPU-based parallel simulations.

PBRL framework is successfully applied to both off-policy (SAC, DDPG) and on-policy (PPO) RL algorithms, analyzing the implications of critical design choices, i.e., the number of agents and the mutation mechanisms.

1.1.3 Sim-to-Real Transfer

Despite the calibration efforts to model the physical system accurately, simulation is still a rough approximation. The differences between the dynamics of simulated and real systems cause a "reality gap" that makes it unlikely for a simulation-trained policy to successfully transfer to a physical system. In literature, researchers have put a significant effort into diminishing this gap: to this aim, most of the approaches leverage domain randomization [24, 4, 3, 27, 8, 7] to expose the policy to a wide range of observation distributions in simulation, thus improving generalization onto a real system. Nevertheless, naive domain randomization might not be sufficient to completely attenuate the dynamics gap: for instance, [13] employs a specific network to mimic the real actuation system. Another technique in this context is policy-level action integrator (PLAI) [32], a simple yet effective algorithm aimed at compensating the sim-to-real dynamic discrepancies with an integral action, which proved to be paramount for a successful transfer.

In this paper, we employ sim-to-real strategies to deploy a policy on a real system; to the best of the authors' knowledge, this work represents the first instance of deploying PBRL agents on real hardware.

1.2 Contribution

This paper investigates a population-based reinforcement learning (PBRL) framework in robotics that allows the training of a population of agents by exploiting GPU-based massively parallel simulation to dynamically adjust the hyperparameters during training. We evaluate the PBRL framework on four complex tasks that require learning essential skills for humanoid robots: *Anymal Terrain*, *Humanoid*, *Shadow Hand*, and *Franka Nut Pick* (Figure 1), available in Isaac Gym [21]. The results show that better performance is achieved when training a population of agents compared to a single-agent baseline on all tasks. The comparison is provided across 3 RL algorithms (PPO, SAC, and DDPG), varying the number of agents in a population, and across different hyperparameter mutation schemes. Finally, the PBRL agents are deployed on a real Franka Panda robot for a *Franka Nut Pick* task, without any policy adaptation phase on the physical system. In summary, the main contributions of this work are:

- a population-based RL framework that utilizes GPU-accelerated simulation to train robotic manipulation tasks by adaptively optimizing the set of hyperparameters during training;
- simulations demonstrating the effectiveness of the PBRL approach on 4 tasks using 3 RL algorithms, including both on-policy and off-policy methods, investigating the performance w.r.t. the number of agents and mutation mechanisms;
- sim-to-real transfer of PBRL policies onto a real Franka Panda robot;

- an open-source codebase to train policies using the PBRL algorithm.

2 Methods

This section describes the core concepts involved in the PBRL framework. The overall approach, illustrated in Figure 2, can be viewed as a multi-layered training process consisting of an inner optimization loop with RL and an outer loop of online evolutionary selection with population-based training. During training, the parameters of the agent's policy are updated at a higher rate using RL than the hyperparameters defining the RL procedure.

2.1 Reinforcement Learning

The RL problem is modeled as a Markov Decision Process (MDP), where an agent interacts with the environment in order to maximize the expected sum of episodic rewards. Specifically, an MDP is defined as $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \gamma)$, where $\mathcal{S}$ is the set of states, $\mathcal{A}$ is the set of actions, $\mathcal{T}$ is the transition dynamics, i.e., $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \mathbb{P}(\mathcal{S})$, where $\mathbb{P}(\mathcal{S})$ defines the set of a probability distribution over $\mathcal{S}$, $\mathcal{R} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function, and $\gamma \in [0, 1]$ represents the discount factor. The goal is formulated as learning a policy, either stochastic, $\pi_\theta : \mathcal{S} \to \mathcal{D}_\mathcal{A}$, or deterministic, $\pi_\theta : \mathcal{S} \to \mathcal{A}$, where $\mathcal{D}_\mathcal{A}$ represents a probability distribution over $\mathcal{A}$ and θ encapsulates the policy parameters, whose cardinality depends on the selected algorithm and network architecture. In this work, the policy is learned using the on-policy method PPO, or either of the off-policy methods SAC or DDPG. All these algorithms use an actor-critic architecture simultaneously learning the policy (actor) and the value function approximators (critics) $\mathcal{Q} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$. The implementation of critics in SAC and DDPG relies on double Q-learning and n-step returns.

To train the policy with PPO, a learning rate (LR) adaptation procedure is used based on a Kullback–Leibler (KL) divergence starting from an initial value η_0 [21]. At the end of each update iteration, the LR is increased by a factor of K_η when the KL divergence between the current policy and the old policy is below the specified threshold, or reduced by K_η if the KL divergence exceeds the threshold.

In DDPG, the common practice involves adding a small noise to the deterministic actions of the policy to enable exploration. In this work, the noise is added following a mixed exploration strategy [18], where the general idea is akin to adding a different noise level for each environment when training in a massively parallel regime. For the i-th environment out of $N \in \mathbb{Z}^+$ environments, a zero mean and uncorrelated Gaussian noise is given as: $\mathcal{N}(0, \sigma_i)$, where $\sigma_i = \sigma_{min} + \frac{i-1}{N-1}(\sigma_{max} - \sigma_{min})$.

2.2 Population-Based Training

In standard RL, the agent aims to learn an optimal policy by interacting with an environment and iteratively updating the policy through some kind of optimization method. In contrast, PBRL uses a population of n agents $\mathcal{P}$, each interacting with the environment independently to collect experience and learn its own policy. Using evolutionary selection, the population is periodically evaluated based on a fitness metric, and best-performing members replace the worst-performing members, i.e., weights of the best agents are copied over, along with the mutated hyperparameters.

In this work, a specific PBRL approach, population-based training (PBT), is employed as an outer optimization loop to enable diverse exploration and dynamically adapt the hyperparameters in high-dimensional continuous control tasks. Each agent $\pi(\theta_i, h_i) \in \mathcal{P}$ is characterized by a vector θ_i and the set of hyperparameters h_i, where θ_i contains the parameters of the policy, and h_i contains the hyperparameters that are optimized during training. To represent the whole population $\mathcal{P}$, we denote with $\Theta \triangleq \bigcup_{i=1}^{n} \theta_i$, $\boldsymbol{h} \triangleq [h_1, h_2, \dots, h_n]$ and $\Pi \triangleq \{\pi(\theta_i, h_i)\}_{i=1}^{n}$ the sets of all the parameters, hyperparameters and policies respectively.

Algorithm 1 provides pseudocode for the PBRL. The training process runs in iterations, where all agents are first independently trained by performing updates to the vector θ_i. After a certain number of policy updates N_{evo} (each agent having been trained for some steps), the agents are evaluated and sorted based on the average return $\mathcal{R}_{mean}$ obtained over all of the previous episodes. The agents in $\mathcal{P}_{bottom\,25\,\%}$ get replaced by randomly-sampled agents in $\mathcal{P}_{top\,25\,\%}$ with mutated hyperparameters, while the rest of the agents in $\mathcal{P}_{mid\,50\,\%}$ and $\mathcal{P}_{top\,25\,\%}$ continue training.

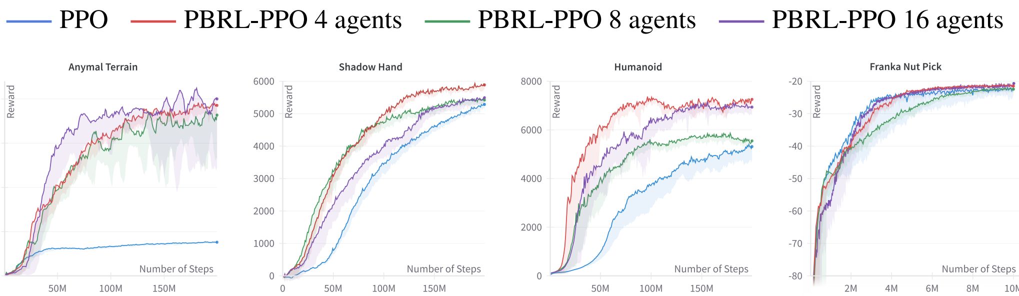


Figure 3: Training results of baseline PPO and PBRL-PPO for $|\mathcal{P}| \in \{4, 8, 16\}$. The shaded area represents the performance between the best and the worst agent in $\mathcal{P}$, or among 4 different seeds in a non-evolutionary baseline.

To generate the mutated hyperparameters, 3 mutation mechanisms are considered (see line 14 of Algorithm 1): (i) random perturbation is applied to the hyperparameters of the parent agent(s) through perturbation factors in Table 5; (ii) new hyperparameters are sampled from a prior uniform distribution with bounds specified in Table 3 and 4; (iii) according to the DexPBT mutation scheme [26], hyperparameters are multiplied or divided by a random number μ sampled from a uniform distribution, i.e., $\mu \sim \mathcal{U}(\mu_{min}, \mu_{max})$ with probability $\beta_{mut} \in [0, 1]$. Section 3.2.4 compares all 3 mutation schemes. After beginning the training, evolution is enabled after $N_{start} \in \mathbb{Z}^+$ steps as in [14] to allow for initial exploration and promote population diversity.

3 Experiments

3.1 Environments

The PBRL framework is evaluated on some of the most challenging benchmark tasks available in Isaac Gym, including *Anymal Terrain*, *Shadow Hand*, *Humanoid* and *Franka Nut Pick* (Figure 1). The experiments are conducted on a workstation with a single NVIDIA RTX 4090 GPU and 32 GB of RAM. Parallelizing the data collection across the GPU, Isaac Gym's PhysX engine can simulate thousands of environments using the above hardware.

3.2 Results

The experiments focus on optimizing the hyperparameters of the RL agents in a population and comparing the results against non-evolutionary baseline agents. For each case of baseline agents, 4 experiments are run with different seeds. Tables 3 and 4 list the hyperparameters for on-policy and off-policy algorithms, including the sampling ranges of those optimized through the PBRL Algorithm 1. The initial values for each agent are uniformly sampled from a prior distribution with a given range.

3.2.1 PBRL-PPO

For the PPO agents, the tuned hyperparameters are the KL divergence threshold for an adaptive LR, the entropy loss coefficient, and the variance of action selection. These parameters are crucial in ensuring sufficient exploration of the environment. Figure 3 shows the learning curves for the single-agent PPO baseline and PBRL-PPO for $|\mathcal{P}| \in \{4, 8, 16\}$. The results demonstrate that PBRL-PPO outperforms PPO on 3 out of 4 tasks, yielding a higher return, with significant improvement seen in *Anymal Terrain*, which involves traversing increasingly challenging terrain. For *Franka Nut Pick*, PBRL agents achieve comparable performance to the baseline PPO agents. This is because, in this relatively straightforward task, randomization alone suffices for a thorough exploration of the state/action space.

3.2.2 PBRL-SAC

In PBRL-SAC, the optimized hyperparameters include the LR of the actor-critic networks and the target entropy factor. Entropy is key in SAC agents as the policy is trained to maximize the tradeoff between the expected return and exploration. Experiments are run with a population size of $|\mathcal{P}| \in \{4, 8\}$. Due to higher memory needs for replay buffers in off-policy methods, the maximum population size is limited to 8. The training performance of SAC and PBRL-SAC is shown in Figure 4. PBRL-SAC improves the training performance compared to non-evolutionary SAC on 3 out of 4 tasks, yielding a remarkable improvement on both *Shadow Hand* and *Franka Nut Pick*, while comparable results are achieved on *Humanoid*, probably due to the limited task complexity.

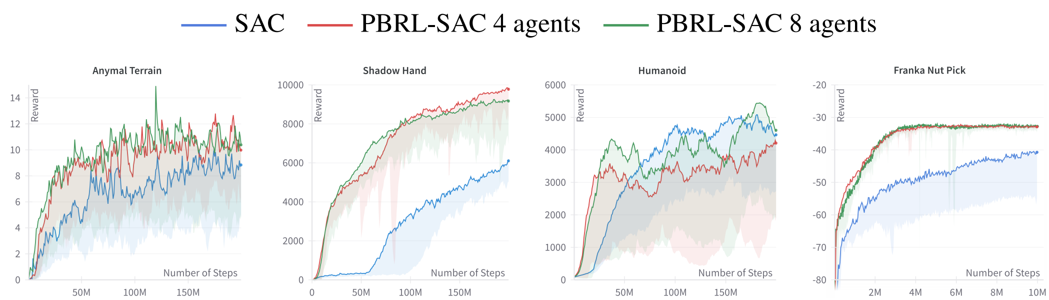


Figure 4: Training results of baseline SAC and PBRL-SAC for $|\mathcal{P}| \in \{4, 8\}$. The shaded area displays the performance between the best and the worst agent in $\mathcal{P}$, or among 4 different seeds in a non-evolutionary baseline.

3.2.3 PBRL-DDPG

In DDPG, exploration noise is added to the output of the deterministic actor. As mentioned in Section 2.1, different noise levels are added for different environments uniformly within the range $[\sigma_{min}, \sigma_{max}]$. Both these parameters are crucial in controlling the amount of exploration in DDPG agents. In PBRL-DDPG, the hyperparameters optimized during training include the minimum and the maximum bounds for noise levels, i.e., $\sigma_{min}, \sigma_{max}$, and the LRs of the actor and the critic. As in PBRL-SAC, the maximum population size in PBRL-DDPG is set to 8 due to the presence of independent replay buffers and GPU memory limitations. Figure 5 shows that PBRL-DDPG achieves significantly better training performance than DDPG on all 4 benchmark tasks.

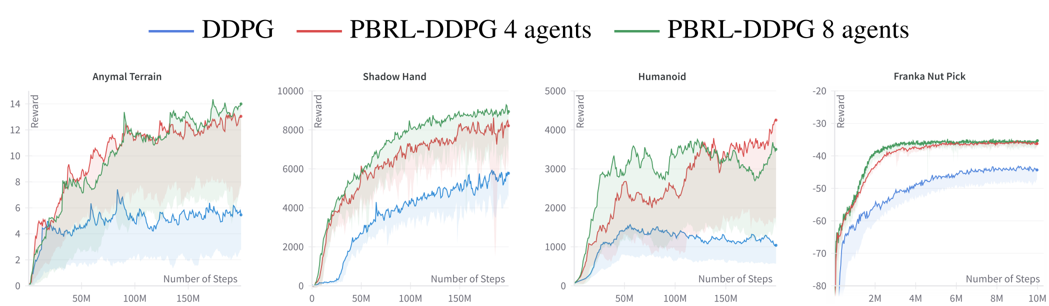


Figure 5: Training results of baseline DDPG and PBRL-DDPG for $|\mathcal{P}| \in \{4, 8\}$. The shaded area displays the performance between the best and the worst agent in $\mathcal{P}$, or among 4 different seeds in a non-evolutionary baseline.

3.2.4 Mutation Comparison

Figure 6 shows the results using 3 different mutation schemes for PBRL-PPO and PBRL-DDPG. As mentioned in Section 2.2, the hyperparameters for under-performing agents are generated either by sampling from an original prior distribution, by perturbing the parent's values through perturbation factors given in Table 5, or through the DexPBT mutation scheme presented in [26]. In the latter, the hyperparameters have a $\beta_{mut} := 0.5$ probability of getting multiplied or divided by a random number sampled from the uniform distribution, $\mu \sim \mathcal{U}(1.1, 1.5)$. The results show that the perturbed agents either exceed or are on par with the performance of other mutation schemes in 6 out of 8 experiments. The DexPBT mutation scheme performs better with PBRL-DDPG on *Humanoid* and *Franka Nut Pick* tasks, which are less challenging compared to others. The combination of two

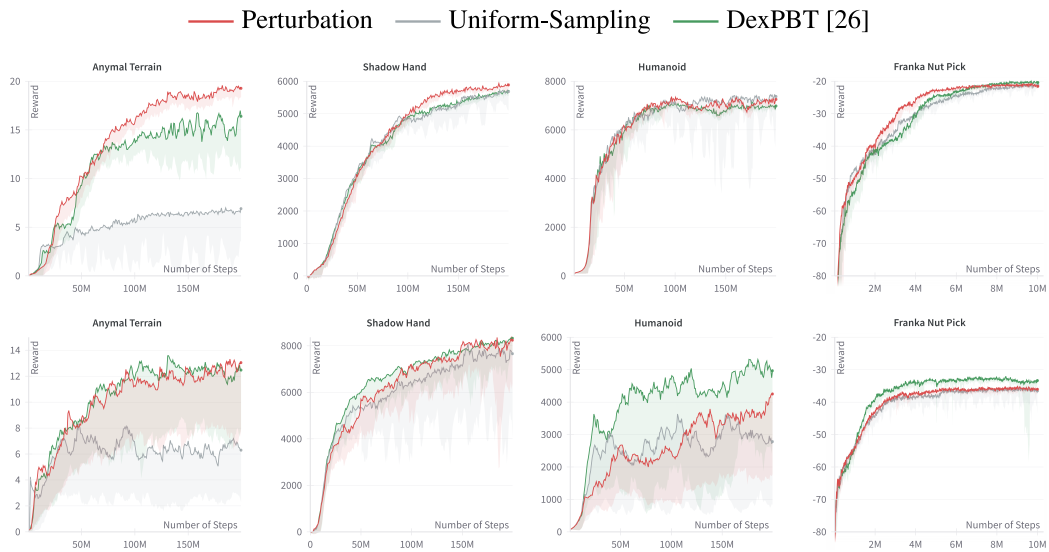


Figure 6: Comparison of different mutation schemes for PBRL-PPO (top) and PBRL-DDPG (bottom) with $|\mathcal{P}| = 4$.

mutation schemes might discover better exploration strategies for a wider range of tasks. Analyzing the potential synergies between the two remains a prospect for future investigation.

3.3 Sim-to-Real Transfer

In the real experiments, we replicate the *Franka Nut Pick* task [24] by deploying a PBRL-PPO policy, without any real-world adaptation, executing the actions with PLAI [32]. The robot detects the nuts utilizing Mask-RCNN [12], fine-tuned on real-world images captured with a wrist-mounted Intel RealSense D435 camera, using the `IndustRealLib` codebase [32]. Compared to the original task [24], we applied the following changes to make the simulated environment resemble real setup: (i) employing a Task-Space Impedance (TSI) controller [6] instead of an Operational-Space motion Controller (OSC) [17] to comply with the actual low-level controller[1]; (ii) randomizing the nut's initial position to reflect the actual robot workspace; (iii) changing the observation space to include the 7-DOF joint configuration, the measured end-effector pose, and the estimated nut pose. The parameters used in the simulated environment and the real controller are reported in Table 2.

During experiments, the following policies were deployed, performing 30 real-world trials of *Franka Nut Pick* task for each policy: (i) 2 agents from a population of 8, trained with PBRL-PPO, specifically the "best" and the "worst" agent; (ii) the "best" agent trained with baseline PPO. With "best" and "worst" we indicate the agents achieving the highest and lowest *success rate* in simulation, where success is defined as reaching, grasping, and lifting the nut, without losing contact during the lifting phase. PBRL-PPO with $|\mathcal{P}| = 8$ achieved the highest success rate. Remarkably, we found out that even the success rate of the worst agent in $\mathcal{P}$ was higher than that of the best PPO agent.

Deploying both PPO and PBRL-PPO agents onto a real robot leads to task completion (shown in Figure 7), yet with different success rates, as summarized in Table 1. Particularly, both PBRL-PPO

[1]The control laws are specified in [24] and in reference works [6, 17]

Table 1: Success rate deploying the best and the worst of 8 agents trained with PBRL-PPO and the best PPO baseline agent on the *Franka Nut Pick* task with the real robot

Algorithm	**Agent**	**Successful trials**	**Success rate**
PBRL-PPO	Best	27/30	90%
PBRL-PPO	Worst	21/30	70%
PPO	Best	19/30	63.33%

Figure 7: Snapshots of the *Franka Nut Pick* experiment on the real robot: full video on our project website.

agents yield higher success rates than PPO, with the “best” agent performing better than the “worst” one, indeed confirming the ranking attained in simulation. Unlike the baseline PPO agent, which continued to produce small movements after reaching the target, PBRL-PPO agents remained more stable, leading to a higher success rate. This demonstrates that PBRL agents, while achieving similar rewards to a single agent, learn behaviors that exhibit greater robustness to environment variability due to the diversity in agent populations. Informally, the best PBRL-PPO agent also exhibited recovery behavior during task execution after perturbation by the human.

3.4 Discussion

While the PBRL agents perform better than the non-evolutionary agents in almost all the experiments, the impact of population size across RL algorithms and tasks shows no consistent pattern. One may hypothesize that larger and more diverse populations might lead to a better final performance. However, the results in this work indicate that using a larger population size does not necessarily yield substantial benefits for every task. This is in contrast to the common belief that population-based methods rely on larger population sizes to effectively explore the hyperparameter space [15, 25]. The optimal population size, instead, depends on various factors, including task complexity, RL algorithm, and interaction dynamics among agents. While larger populations offer increased exploration potential, they also suffer from diminished exploitation capabilities due to increased competition, leading to lower performance in less challenging tasks where smaller populations suffice. Larger population sizes seem to perform better when the task complexity gradually increases requiring extensive exploration as in *Anymal Terrain*, which implements curriculum learning.

Additionally, the performance of PBRL may be lower than non-evolutionary agents on relatively simpler tasks where optimal hyperparameters are known *a priori*. This can be noticed on a *Humanoid* task trained with SAC in Figure 4: indeed, baseline policies achieve a higher reward than PBRL-SAC with 4 agents; nevertheless, 8 agents are capable of outperforming the baseline. Thus, the benefits provided by PBRL will become more apparent for new tasks where ideal hyperparameter ranges are not known in advance. In this sense, PBRL can be thought of as an exploratory approach to search for unknown optimal configurations of newly designed tasks.

4 Conclusion

In this paper, a PBRL framework has been employed to train a population of RL agents by making use of high-throughput GPU-accelerated simulation. The first simulation results of PBRL using on-policy and off-policy methods are provided on a series of locomotion and manipulation benchmark tasks proposed in [21] by investigating the effect of population size and different mutation schemes. The results showed the effectiveness of PBRL in improving final performance through online adaptation of hyperparameters. PBRL agents have been deployed on real hardware for the first time, demonstrating smooth and successful transfer, without any policy adaptation or fine-tuning. Finally, we released the codebase to train PBRL agents and hope that it will empower researchers to further explore and extend the capabilities of PBRL algorithms.

References

[1] D. Ackley and M. Littman. Interactions between learning and evolution. *Artif. Life II*, 10:487–509, 1991.

[2] I. Akkaya, M. Andrychowicz, M. Chociej, M. Litwin, B. McGrew, A. Petron, A. Paino, M. Plappert, G. Powell, R. Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint: 1910.07113*, 2019.

[3] A. Allshire, M. MittaI, V. Lodaya, V. Makoviychuk, D. Makoviichuk, F. Widmaier, M. Wüthrich, S. Bauer, A. Handa, and A. Garg. Transferring dexterous manipulation from gpu simulation to a remote real-world trifinger. In *Proc. IEEE Int. Conf. Intell. Robots Syst.*, pages 11802–11809, Oct. 2022.

[4] O. M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, et al. Learning dexterous in-hand manipulation. *Int. J. Robot. Res.*, 39(1):3–20, Jan. 2020.

[5] C. Berner, G. Brockman, B. Chan, V. Cheung, P. Debiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint: 1912.06680*, 2019.

[6] F. Caccavale, C. Natale, B. Siciliano, and L. Villani. Six-DOF impedance control based on angle/axis representations. *IEEE Trans. Robot. Automat.*, 15(2):289–300, Apr. 1999.

[7] Y. Chebotar, A. Handa, V. Makoviychuk, M. Macklin, J. Issac, N. Ratliff, and D. Fox. Closing the Sim-to-Real Loop: Adapting Simulation Randomization with Real World Experience. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages 8973–8979, May 2019.

[8] C. Chi, B. Burchfiel, E. Cousineau, S. Feng, and S. Song. Iterative Residual Policy for Goal-Conditioned Dynamic Manipulation of Deformable Objects. In *Proc. Robot. Sci. Syst.*, June 2022.

[9] A. Flajolet, C. B. Monroc, K. Beguir, and T. Pierrot. Fast population-based reinforcement learning on a single machine. In *Proc. Int. Conf. Mach. Learn.*, volume 162, pages 6533–6547, July 2022.

[10] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *arXiv preprint: 1801.01290*, 2018.

[11] A. Handa, A. Allshire, V. Makoviychuk, A. Petrenko, R. Singh, J. Liu, D. Makoviichuk, K. Van Wyk, A. Zhurkevich, B. Sundaralingam, and Y. Narang. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages 5977–5984, June 2023.

[12] K. He, G. Gkioxari, P. Dollar, and R. Girshick. Mask R-CNN. In *Proc. IEEE Int. Conf. Comput. Vis.*, pages 2980–2988, Oct. 2017.

[13] J. Hwangbo, J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun, and M. Hutter. Learning agile and dynamic motor skills for legged robots. *Sci. Robot.*, 4(26), Jan. 2019.

[14] M. Jaderberg, W. M. Czarnecki, I. Dunning, L. Marris, G. Lever, A. G. Castañeda, C. Beattie, N. C. Rabinowitz, A. S. Morcos, A. Ruderman, N. Sonnerat, T. Green, L. Deason, J. Z. Leibo, D. Silver, D. Hassabis, K. Kavukcuoglu, and T. Graepel. Human-level performance in 3D multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, May 2019.

[15] M. Jaderberg, V. Dalibard, S. Osindero, W. M. Czarnecki, J. Donahue, A. Razavi, O. Vinyals, T. Green, I. Dunning, K. Simonyan, et al. Population based training of neural networks. *arXiv preprint*, 2017.

[16] G. Kahn, A. Villaflor, B. Ding, P. Abbeel, and S. Levine. Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation. In *Proc. IEEE Int. Conf. Robot. Automat.*, pages 5129–5136, May 2018.

[17] O. Khatib. A unified approach for motion and force control of robot manipulators: The operational space formulation. *IEEE J. Robot. Automat.*, 3(1):43–53, Feb. 1987.

[18] Z. Li, T. Chen, Z.-W. Hong, A. Ajay, and P. Agrawal. Parallel Q-learning: Scaling off-policy reinforcement learning under massively parallel simulation. In *Proc. Int. Conf. Mach. Learn.*, volume 202, pages 19440–19459, July 2023.

[19] J. Liang, V. Makoviychuk, A. Handa, N. Chentanez, M. Macklin, and D. Fox. GPU-accelerated robotic simulation for distributed reinforcement learning. In *Proc. Conf. Robot Learn.*, volume 87, pages 270–282, Oct. 2018.

[20] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint: 1509.02971*, 2015.

[21] V. Makoviychuk, L. Wawrzyniak, Y. Guo, M. Lu, K. Storey, M. Macklin, D. Hoeller, N. Rudin, A. Allshire, A. Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint: 2108.10470*, 2021.

[22] G. Margolis, G. Yang, K. Paigwar, T. Chen, and P. Agrawal. Rapid Locomotion via Reinforcement Learning. In *Proc. Robot. Sci. Syst.*, June 2022.

[23] T. Miki, J. Lee, J. Hwangbo, L. Wellhausen, V. Koltun, and M. Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Sci. Robot.*, 7(62), Jan. 2022. Art. no. eabk2822.

[24] Y. Narang, K. Storey, I. Akinola, M. Macklin, P. Reist, L. Wawrzyniak, Y. Guo, A. Moravanszky, G. State, M. Lu, A. Handa, and D. Fox. Factory: Fast Contact for Robotic Assembly. In *Proc. Robot. Sci. Syst.*, June 2022.

[25] J. Parker-Holder, V. Nguyen, and S. J. Roberts. Provably efficient online hyperparameter optimization with population-based bandits. In *Proc. Adv. Neural Inform. Process. Syst.*, volume 33, pages 17200–17211, Dec. 2020.

[26] A. Petrenko, A. Allshire, G. State, A. Handa, and V. Makoviychuk. DexPBT: Scaling up Dexterous Manipulation for Hand-Arm Systems with Population Based Training. In *Proc. Robot. Sci. Syst.*, July 2023.

[27] N. Rudin, D. Hoeller, P. Reist, and M. Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning. In *Proc. Conf. Robot Learn.*, volume 164, pages 91–100, Nov. 2022.

[28] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms. *arXiv preprint: 1707.06347*, 2017.

[29] A. A. Shahid, D. Piga, F. Braghin, and L. Roveda. Continuous control actions learning and adaptation for robotic manipulation through reinforcement learning. *Auton. Robots*, 46(3):483–498, Feb. 2022.

[30] A. A. Shahid, J. S. V. Sesin, D. Pecioski, F. Braghin, D. Piga, and L. Roveda. Decentralized multi-agent control of a manipulator in continuous task learning. *Appl. Sci.*, 11(21):Art. no. 10227, Nov. 2021.

[31] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, Dec. 2018.

[32] B. Tang, M. A. Lin, I. A. Akinola, A. Handa, G. S. Sukhatme, F. Ramos, D. Fox, and Y. S. Narang. IndustReal: Transferring Contact-Rich Assembly Tasks from Simulation to Reality. In *Proc. Robot. Sci. Syst.*, July 2023.

[33] E. Todorov, T. Erez, and Y. Tassa. MuJoCo: A physics engine for model-based control. In *Proc. IEEE Int. Conf. Intell. Robots Syst.*, pages 5026–5033, Oct. 2012.

[34] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, Nov. 2019.

[35] J. Xu, V. Makoviychuk, Y. Narang, F. Ramos, W. Matusik, A. Garg, and M. Macklin. Accelerated policy learning with parallel differentiable simulation. In *Proc. Int. Conf. Learn. Represent.*, Apr. 2022. Art. no. 186704.

A Algorithm

Algorithm 1 PBRL algorithm

Require: Initial population $\mathcal{P}$ of agents (Θ random, $\boldsymbol{h}$ sampled from a uniform distribution)

$N_{iter} = 0$
while not end of training **do**
  $\theta \leftarrow$ Train($\Pi(\Theta, \boldsymbol{h})$) ▷ Train all agents in $\mathcal{P}$
  $N_{iter} = N_{iter} + 1$
  if $N_{iter} > N_{start}$ and $N_{iter} \ \% \ N_{evo} = 0$ **then**
    for each agent $\pi(\theta, h) \in \mathcal{P}$ **do**
      $\mathcal{R}_{mean} \leftarrow$ Eval($\pi(\theta, h)$)
      Sort $\pi(\theta, h)$ based on $\mathcal{R}_{mean}$
    end for
    Partition $\mathcal{P}$ into $\mathcal{P}_{top\,25\,\%}$, $\mathcal{P}_{mid\,50\,\%}$, $\mathcal{P}_{bottom\,25\,\%}$
    Sample $\pi^*(\theta^*, h^*)$ from $\mathcal{P}_{top\,25\,\%}$ at random
    for each agent $\pi(\theta, h) \in \mathcal{P}_{bottom\,25\,\%}$ **do**
      $\pi(\theta) \leftarrow \pi^*(\theta^*)$
      $h \leftarrow$ Mutate(h^*)
    end for
  end if
end while

B Domain Randomization for Franka Nut Pick Task

In this section, we include the settings used for domain randomization in experiments with the Franka robotic arm for simulated environment and real setup. The robot initial pose is randomized according to a Gaussian distribution $\mathcal{N}$, while the nut initial position is uniformly chosen in the specified range.

Table 2: Simulated environment and real control configuration parameters used in *Franka Nut Pick* during training and deployment respectively.

Parameter	**Value**
Franka initial position	$\mathcal{N}([0.0, -0.2, 0.2], [0.2, 0.2, 0.1])$
Franka initial rotation	$\mathcal{N}([\pi, 0, \pi], [0.3, 0.3, 1])$
Nut initial position	[0.42, 0.27, 0.02] ± [0.18, 0.13, 0.01]
TSI proportional gains	[1000, 1000, 1000, 50, 50, 50]
TSI derivative gains	[63.25, 63.25, 63.25, 1.414, 1.414, 1.414]
Action scale	0.0001

C Hyperparameters

Table 3: Hyperparameters setup for PPO and PBRL-PPO across all the tasks.

Hyperparameter	**PPO**			**PBRL-PPO**		
	Anymal Terrain	*Shadow Hand & Humanoid*	*Franka Nut Pick*	*Anymal Terrain*	*Shadow Hand & Humanoid*	*Franka Nut Pick*
Environments per agent	4096	16384	128	1024	4096	128
MLP hidden units	[512, 256, 128]	[512, 256, 128]	[256, 128, 64]	[512, 256, 128]	[512, 256, 128]	[256, 128, 64]
Horizon	32	16	120	32	16	120
Batch size	8192	32768	512	8192	8192	512
Actor variance	0.5	1	1	0.3 – 1	0.3 – 1	0.3 – 1
KL threshold	0.016	0.016	0.016	0.08 – 0.016	0.08 – 0.016	0.08 – 0.016
Entropy loss coefficient	0.001	0.001	0	0.0001 – 0.001	0.0001 – 0.001	0.0001 – 0.001
Epochs	8	4	8	8	4	8
Discount factor γ	0.99	0.99	0.99	0.99	0.99	0.99
GAE lambda	0.95	0.95	0.95	0.95	0.95	0.95
PPO clip ϵ	0.2	0.2	0.2	0.2	0.2	0.2
Initial LR η_0	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}	5×10^{-4}
LR adaptation gain K_η	1.5	1.5	1.5	1.5	1.5	1.5

Table 4: Hyperparameters setup for off-policy algorithms on all four tasks. [*]For *Franka Nut Pick* these parameters are, respectively: 128, [256, 128, 64], 512.

Hyperparameter	**SAC & DDPG**	**PBRL-SAC & PBRL-DDPG**
Environments per agent[*]	2048	2048
MLP hidden units[*]	[512, 256, 128]	[512, 256, 128]
Batch size[*]	4096	4096
Horizon	1	1
Target update rate τ	5×10^{-2}	5×10^{-2}
Actor learning rate	0.0001	0.0001 – 0.001
Critic learning rate	0.0001	0.0001 – 0.001
DDPG exploration σ_{min}	0.01	0.01 – 0.1
DDPG exploration σ_{max}	1	0.5 – 1
SAC target entropy	-20	-20 – -10
Replay buffer size	1×10^6	1×10^6
Epochs	4	4
n-step returns	3	3

Table 5: Parameter setup for PBRL

Parameter	**Value**	
	Franka Nut Pick	Others
Evolution start N_{start}	2×10^5 steps	1×10^7 steps
Evolution frequency N_{evo}	1×10^5 steps	2×10^6 steps
Perturbation factor (min.)	0.8	0.8
Perturbation factor (max.)	1.2	1.2