SAPG: Split and Aggregate Policy Gradients

Jayesh Singla [*1] **Ananye Agarwal** [*1] **Deepak Pathak** [1]

Carnegie Mellon University

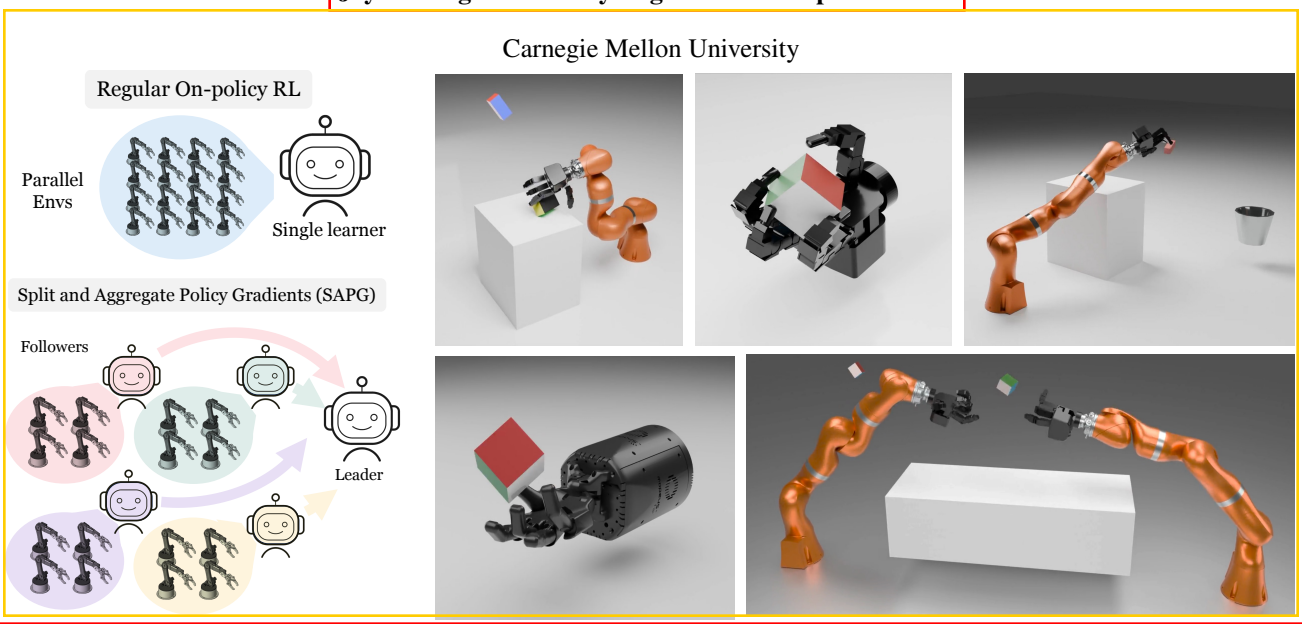

Figure 1. We introduce a new class of on-policy RL algorithms that can scale to tens of thousands of parallel environments. In contrast to regular on-policy RL, such as PPO, which learns a single policy across environments leading to wasted environment capacity, our method learns diverse followers and combines data from them to learn a more optimal leader in a continuous online manner.

Abstract

Despite extreme sample inefficiency, on-policy reinforcement learning, aka policy gradients, has become a fundamental tool in decision-making problems. With the recent advances in GPU-driven simulation, the ability to collect large amounts of data for RL training has scaled exponentially. However, we show that current RL methods, e.g. PPO, fail to ingest the benefit of parallelized environments beyond a certain point and their performance saturates. To address this, we propose a new on-policy RL algorithm that can effectively leverage large-scale environments by splitting them into chunks and fusing them back together via importance sampling. Our algorithm, termed SAPG, shows significantly higher performance across a variety of challenging environments where vanilla PPO and other strong baselines fail to achieve high performance. Webpage at https://sapg-rl.github.io.

1. Introduction

Broadly, there are two main categories in reinforcement learning (RL): off-policy RL, e.g., Q-learning [32], and on-policy RL, e.g., policy gradients [29]. On-policy methods are relatively more sample inefficient than off-policy but often tend to converge to higher asymptotic performance. Due to this reason, on-policy RL methods, especially PPO [27], are usually the preferred RL paradigm for almost all sim2real robotic applications [19, 1, 2] to games such as StarCraft [30], where one could simulate years of real-world experience in minutes to hours.

RL is fundamentally a trial-n-error-based framework and hence is sample inefficient in nature. Due to this, one needs to have large batch sizes for each policy update, especially in the case of on-policy methods because they can only use data from current experience. Fortunately, in recent years, the ability to simulate a large number of environments in parallel has become exponentially larger due to GPU-driven physics engines, such as IsaacGym [17], PhysX, Mujoco-3.0, etc. This means that each RL update can easily scale to batches of size hundreds of thousands to millions, which are over two orders of magnitude higher than what most RL

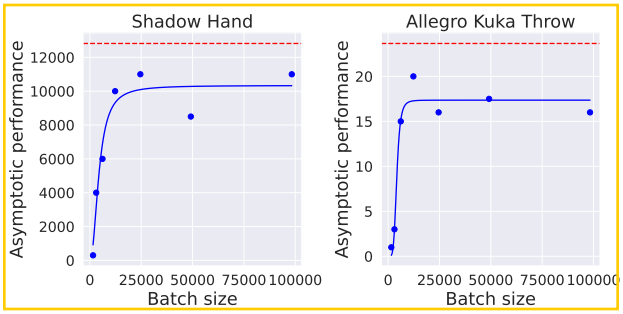

Figure 2. Performance vs batch size plot for PPO runs (blue curve) across two environments. The curve shows how PPO training runs can not take benefit of large batch size resulting from massively parallelized environments and their asymptotic performance saturates after a certain point. The dashed red line is the performance of our method, SAPG, with more details in the results section. It serves as evidence that higher performance is achievable with larger batch sizes.

benchmarks typically have.

In this paper, we highlight an issue with typical on-policy RL methods, e.g. PPO, that they are not able to ingest the benefits with increasingly larger sample sizes for each update. In Figure 2, we show that PPO performance saturates after a certain batch size despite the ceiling being higher. This is due to the issue in data sampling mechanisms. In particular, at each timestep actions are sampled from a Gaussian with some mean and variance. This implies that most sampled actions are near the mean and with large number of environments, many environments are executing the same actions leading to duplicated data. This implies that the performance of PPO saturates at some point as we increase the number of environments.

We propose a simple fix to this problem. Instead of running a single PPO policy for all environments, we divide environments into blocks. Each block optimizes a separate policy, allowing for more data diversity than just i.i.d. sampling from the same Gaussian. Next, we do an off-policy update to combine data from all these policies to keep the update consistent with the objective of on-policy RL. This allows us to use the PPO's clipped surrogate objective, maintaining the stability benefits of PPO while latching onto high reward trajectories even though they are off-policy. A schematic of our approach, termed SAPG, is shown in Figure 1. We evaluate SAPG across a variety of environments and show significantly high asymptotic performance in environments where vanilla PPO even fails to get any positive success.

2. Related Work

Policy gradients REINFORCE [33], one of the earliest policy gradient algorithms uses an estimator of the objective using simple Monte Carlo return values. Works such as [14] and [28] improve the stability of policy gradient algorithms by employing a baseline to decrease the variance of the estimator while not compromising on the bias. [26, 27] incorporate conservative policy updates into policy gradients to increase the robustness.

Distributed reinforcement learning Reinforcement learning algorithms are highly sample inefficient, which calls for some form of parallelization to increase the training speed. When training in simulation, this speed-up can be achieved by distributing experience collection or different parts of training across multiple processes.[22, 21, 6, 12] However, through the introduction of GPU-based simulators such as IsaacGym [17], the capacity of simulation has increased by two to three orders of magnitude. Due to this, instead of focusing on how to parallelize parts of the algorithm, the focus has shifted to finding ways to efficiently utilize the large amount of simulation data. Previous works such as [1, 2, 8, 25, 10] use data from GPU-based simulation to learn policies in complex manipulation and locomotion settings. However, most of these works still use reinforcement learning algorithms to learn a single policy, while augmenting training with techniques like teacher-student-based training and game-based curriculum. We find that using the increased simulation capacity to naively increase the batch size is not the best way to utilize massively parallel simulation.

[24] develop a population-based training framework that divides the large number of environments between multiple policies and using hyperparameter mutation to find a set of hyperparameters that performs well. However, even this does not utilize all the data completely as each policy learns independently. We propose a way to ensure most of the data from the environments contributes to learning by using all collected transitions for each update.

Off-policy Policy Gradients Unlike on-policy algorithms, off-policy algorithms can reuse all collected data or data collected by any policy for their update. Most off-policy algorithms [20, 16, 9] try to learn a value function which is then implicitly/explicitly used to learn a policy. [15] developed a variant of Deep Deterministic Policy Gradient (DDPG) called PQL which splits data collection and learning into multiple processes and shows impressive performance on many benchmark tasks. We use PQL as one of our baselines to compare our method to off-policy RL in complex tasks. Although off-policy algorithms are much more data-efficient, they usually get lower asymptotic performance than on-policy policy gradients. This has inspired works to develop techniques to use off-policy data in on-policy methods. [11] has been one of the major techniques used to realize this. Previous works [5, 31, 6, 7] develop techniques to use off-policy data in on-policy algorithms using importance sampling-based updates along with features

such as bias correction.

3. Preliminaries

In this paper, we propose a modification to on-policy RL to achieve higher performance in the presence of large batch sizes. We build upon PPO, although our proposed ideas are generally applicable to any on-policy RL method.

On-policy RL Let $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho, \gamma)$ be an MDP where $\mathcal{S}$ is the set of states, $\mathcal{A}$ the set of actions, $\mathcal{P}$ are transition probabilities, r the reward function, ρ the initial distribution of states and γ the discount factor. The objective in reinforcement learning is to find a policy $\pi(a|s)$ which maximises the long term discounted reward $\mathcal{J}(\pi) = \mathbb{E}_{s_0\sim\rho, a_t\sim\pi(\cdot|s_t)}\left[\sum_{t=0}^{T-1}\gamma^t r(s_t, a_t)\right]$.

Policy-gradient algorithms [33, 14, 26, 21] optimize the policy using gradient descent with Monte Carlo estimates of the gradient

$$\nabla_\theta J(\pi_\theta) = \mathop{\mathbb{E}}_{s\sim\rho_d, a\sim\pi(\cdot|s)}\left[\nabla_\theta \log(\pi_\theta(a))\hat{A}^{\pi_\theta}(s,a)\right] \quad (1)$$

where $\hat{A}^{\pi_\theta}(s,a)$ is an advantage function that estimates the contribution of the transition to the gradient. A common choice is $\hat{A}^{\pi_\theta}(s,a) = \hat{Q}^{\pi_\theta}(s,a) - \hat{V}^{\pi_\theta}(s)$, where $\hat{Q}^{\pi_\theta}(s,a)$, $\hat{V}^{\pi_\theta}(s)$ are estimated Q and value functions. This form of update is termed as an actor-critic update [14]. Since we want the gradient of the error with respect to the current policy, only data from the current policy (on-policy) data can be utilized.

PPO Actor critic updates can be quite unstable because gradient estimates are high variance and the loss landscape is complex. An update step that is too large can destroy policy performance. Proximal Policy Optimization (PPO) modifies Eq. 1 to restrict updates to remain within an approximate "trust region" where there is guaranteed improvement [26, 13].

$$L_{\text{on}}(\pi_\theta) = \mathop{\mathbb{E}}_{\pi_{\text{old}}}[\min(r_t(\pi_\theta), \text{clip}\left(r_t(\pi_\theta), 1-\epsilon, 1+\epsilon\right))A_t^{\pi_{old}}] \quad (2)$$

Here, $r_t(\pi_\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{old}(a_t|s_t)}$, ϵ is a clipping hyperparameter and π_{old} is the policy collecting the on-policy data. The clipping operation ensures that the updated π stays close to π_{old}. Empirically, given large numbers of samples, PPO achieves high performance, is stable and robust to hyperparameters. However, it was developed for relatively small batch sizes (≈ 100 parallel envs). We find that in the large-scale setting ($>$10k envs), it is suboptimal because many parallel envs are sampling nearly identical on-policy data.

4. Split and Aggregate Policy Gradients

Policy gradient methods are highly sensitive to the variance in the estimate of gradient. Since CPU-based simulators typically run only 100s of environments in parallel, conventional wisdom is to simply sample on-policy data from a Gaussian policy in all the environments since as the number of datapoints increases, the Monte Carlo estimate becomes more accurate. However, this intuition no longer holds in the extremely large-scale data setting where we have hundreds of thousands of environments on GPU-accelerated simulators like IsaacGym. IID sampling from a Gaussian policy will lead to most actions lying near the mean, and most environments will execute similar actions, leading to wasted data (fig. 2).

We propose to efficiently use large numbers of N environments using a divide-and-conquer setup. Our algorithm trains a variety of M policies $\pi_1, \ldots, \pi_M$ instead of having just one policy. However, simply training multiple policies by dividing environments between them is also inefficient. This is equivalent to training an algorithm with different seeds and choosing the best seed. One approach is to add hyperparameter mutation [24] to the policies and choosing the hyperparameters that perform the best among all of them. However, even in this case, all of the data from the "worse" policies goes to waste, and the only information gained is that some combinations of hyperparameters are bad, even though the policies themselves may have discovered high reward trajectories. We need to somehow aggregate data from multiple policies into a single update. We propose to do this via off-policy updates.

4.1. Aggregating data using off-policy updates

One of the major drawbacks of on-policy RL is its inability to use data from past versions of the policy. One solution is to use importance sampling [5, 18] to weight updates using data from different policies. In practice, this is not used since given limited compute it is beneficial to sample on-policy experience that is more directly relevant. However, this is no longer true in the large batch setting where enough on-policy data is available. In this case, it becomes advantageous to have multiple policies $\pi_1, \ldots, \pi_M$ and use them to sample diverse data, even if it is off-policy. In particular, to update policy π_i using data from policy π_j, $j \in \mathcal{X}$ we use [18]

$$L_{\text{off}}(\pi_i; \mathcal{X}) = \frac{1}{|\mathcal{X}|}\sum_{j\in\mathcal{X}} \mathop{\mathbb{E}}_{(s,a)\sim\pi_j}[\min\left(r_{\pi_i}(s,a), \text{clip}\left(r_{\pi_i}(s,a), \mu(1-\epsilon), \mu(1+\epsilon)\right)\right) A^{\pi_{i,\text{old}}}(s,a)] \quad (3)$$

where $r_{\pi_i}(s,a) = \frac{\pi_i(s,a)}{\pi_j(s,a)}$ and μ is an off-policy correction term $\mu = \frac{\pi_{i,old}(s,a)}{\pi_j(s,a)}$. Note that when $i = j$, then $\pi_j = \pi_{i,old}$ and this reduces to the on-policy update as expected. This

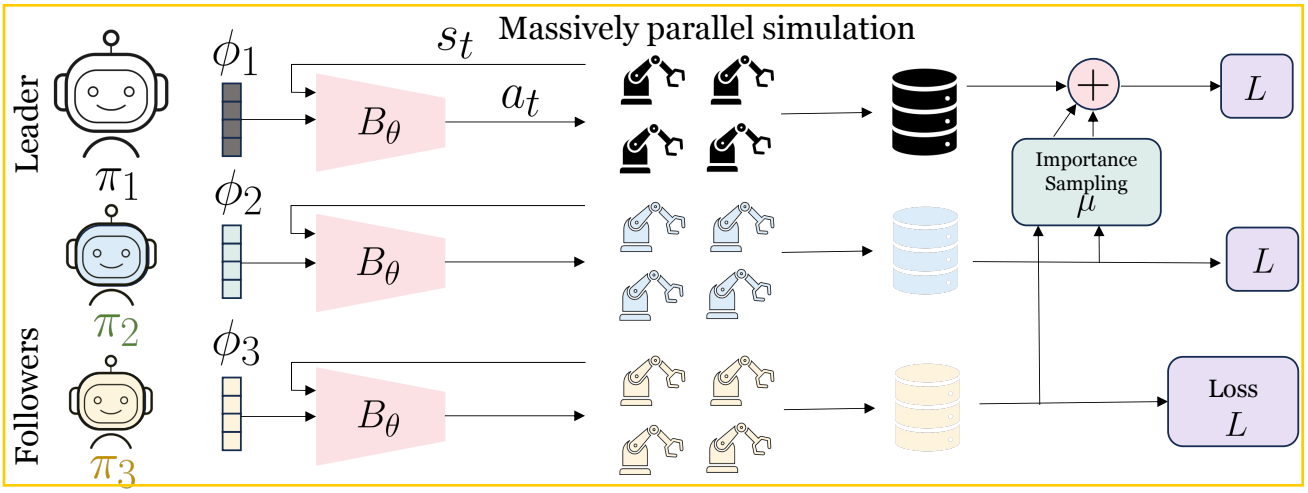

Figure 3. We illustrate one particular variant of SAPG which performs well. There is one leader and $M-1$ followers ($M=3$ in figure). Each policy has the same backbone with shared parameters B_θ but is conditioned on local learned parameters ϕ_i. Each policy gets a block of $\frac{N}{M}$ environments to run. The leader is updated with its on-policy data as well as importance-sampled off-policy data from the followers. Each of the followers only uses their own data for on-policy updates.

is then scaled and combined with the on-policy term (eq. 2)

$$L(\pi_i) = L_{on}(\pi_i) + \lambda \cdot L_{off}(\pi_i; \mathcal{X}) \tag{4}$$

The update target for the critic is calculated using n-step returns (here $n=3$).

$$V_{on,\pi_j}^{target}(s_t) = \sum_{k=t}^{t+2} \gamma^{k-t} r_k + \gamma^3 V_{\pi_j,old}(s_{t+3}) \tag{5}$$

However, this is not possible for off-policy data. Instead, we assume that an off-policy transition can be used to approximate a 1-step return. The target equations are as follows -

$$V_{off,\pi_j}^{target}(s'_t) = r_t + \gamma V_{\pi_j,old}(s'_{t+1}) \tag{6}$$

The critic loss is then

$$L_{on}^{critic}(\pi_i) = \mathop{\mathbb{E}}_{(s,a)\sim\pi_i}\left[(V_{\pi_i}(s) - V_{on,\pi_i}^{target}(s))^2\right] \tag{7}$$

$$L_{off}^{critic}(\pi_i; \mathcal{X}) = \frac{1}{|\mathcal{X}|}\sum_{j\in\mathcal{X}} \mathop{\mathbb{E}}_{(s,a)\sim\pi_j}\left[(V_{\pi_i}(s) - V_{off,\pi_i}^{target}(s))^2\right] \tag{8}$$

$$L^{critic}(\pi_i) = L_{on}^{critic}(\pi_i) + \lambda \cdot L_{off}^{critic}(\pi_i) \tag{9}$$

Given this update scheme, we must now choose a suitable $\mathcal{X} \subseteq \{1,\dots,M\}$ and the set of is to update, along with the correct ratio λ. We explore several variants below.

4.2. Symmetric aggregation

A simple choice is to update all $i's$ with the data from all policies. In this case, we choose to update each policy $i \in \{1,2,\dots,M\}$ and for each i use off-policy data from all other policies $\mathcal{X} = \{1,2,i-1,i+1,\dots,M\}$. Since gradients from off-policy data are typically noisier than gradients from on-policy data, we choose $\lambda = 1$, but subsample the off-policy data such that we use equal amounts of on-policy and off-policy data.

4.3. Leader-follower aggregation

While the above choice prevents data wastage, since all the policies are updated with the same data, it can lead to policies converging in behavior, reducing data diversity and defeating the purpose of having separate policies. To resolve this, we break symmetry by designating a "leader" policy $i=1$ which gets data from all other policies $\mathcal{X} = \{2,3,\dots,M\}$ while the rest are "followers" and only use their own on-policy data for updates $\mathcal{X} = \phi$. As before, we choose $\lambda = 1$, but subsample the off-policy data for the leader such that we use equal amounts of on-policy and off-policy data in a mini-batch update.

4.4. Encouraging diversity via latent conditioning

What is the right parameterization for this set of policies? One simple choice is to have a disjoint set of parameters for each with no sharing at all. However, this implies that each follower policy has no knowledge of any other policy whatsoever and may get stuck in a bad local optimum. We mitigate this by having a shared backbone B_θ for each policy conditioned on hanging parameters ϕ_j local to each policy. Similarly, the critic consists of a shared backbone C_ψ conditioned on parameters ϕ_j. The parameters ψ, θ are shared across the leader and all followers and updated with gradients from each objective, while the parameters ϕ_j are only updated with the objective for that particular policy. We choose $\phi_j \in \mathbb{R}^{32}$ for complex environments

while $\phi_j \in \mathbb{R}^{16}$ for the relatively simpler ones.

4.5. Enforcing diversity through entropy regularization

To further encourage diversity between different policies, in addition to the PPO update loss L_{on} we add an entropy loss to each of the followers with different coefficients. In particular, the entropy loss is $\mathcal{H}(\pi(a \mid s))$. The overall loss for the policy i (or the $(i-1)$th follower is $L(\pi_i) = L_{on}(\pi_i) + \lambda_{\text{ent}}(i-1) \cdot \mathcal{H}(\pi(a \mid s))$. The leader doesn't have any entropy loss. Different scales of coefficients produce policies with different explore-exploit tradeoffs. Followers with large entropy losses tend to explore more actions even if they are suboptimal, while those with small coefficients stay close to optimal trajectories and refine them. This leads to a large data coverage with a good mix of optimal as well as diverse trajectories. We treat λ_{ent} as a hyperparameter.

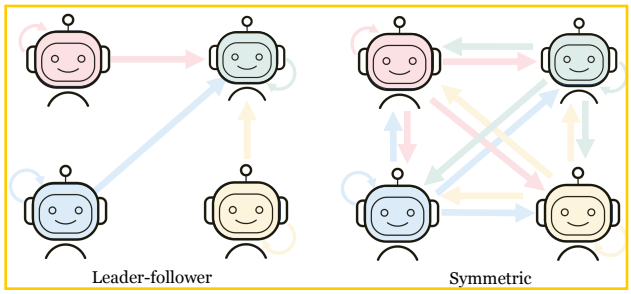

Figure 4. Two data aggregation schemes we consider in this paper. (Left) one policy is a leader and uses data from each of the followers (Right) a symmetric scheme where each policy uses data from all others. In each case, the policy also uses its own on-policy data.

4.6. Algorithm: SAPG

We roll out M different policies and collect data $\mathcal{D}_1, \ldots, \mathcal{D}_M$ for each. Follower policies $2, \ldots, M$ are updated using the usual PPO objective with minibatch gradient descent on their respective datasets. However, we augment the dataset of the leader $\mathcal{D}_1$ with data from $\mathcal{D}_2, \ldots, \mathcal{D}_M$, weighed by the importance weight μ. The leader is then updated by minibatch gradient descent as well.

5. Experimental Setup

We conduct experiments on 5 manipulation tasks (3 hard and 2 easy) and compare them against SOTA methods for the large-scale parallelized setting. We use a GPU-accelerated simulator, IsaacGym [17] which allows simulating tens of thousands of environments in parallel on a single GPU. In our experiments, we focus on the large-scale setting and simulate 24576 parallel environments unless otherwise specified. Note that this is **two orders of magnitude** larger than the number of environments PPO [27] was developed on, and we indeed find that vanilla PPO does not scale to this setting.

Algorithm 1 SAPG

Initialize shared parameters θ, ψ
For $i \in \{1, \ldots, M\}$ initialize parameters ϕ_i
Initialize N environments $E_1, \ldots, E_N$.
Initialize data buffers for each policy $\mathcal{D}_1, \ldots, \mathcal{D}_M$
for $i = 1, 2, \ldots,$ **do**
  for $j = 1, 2, \ldots, M$ **do**
    $\mathcal{D}_j \leftarrow \text{CollectData}\left(E_{j\frac{N}{M}:(j+1)\frac{N}{M}}, \theta, \psi_j\right)$
  end for
  $L \leftarrow 0$
  Sample $|\mathcal{D}_1|$ transitions from $\cup_{j=2}^{M} \mathcal{D}_i$ to get $\mathcal{D}'_1$.
  $L \leftarrow L + \text{OffPolicyLoss}(\mathcal{D}'_1)$
  $L \leftarrow L + \text{OnPolicyLoss}(\mathcal{D}_1)$
  for $j = 2, \ldots, M$ **do**
    $L \leftarrow L + \text{OnPolicyLoss}(\mathcal{D}_j)$
  end for
  Update $\theta \leftarrow \theta - \eta \nabla_\theta L$
  Update $\psi \leftarrow \psi - \eta \nabla_\psi L$
end for

For testing, we choose a suite of manipulation environments that are challenging and require large-scale data to learn effective policies [24]. In particular, these consist of dexterous hands mounted on arms leading to high numbers of degrees of freedom (up to 23). This is challenging because sample complexity scales exponentially with degrees of freedom. They are under-actuated and involve manipulating free objects in certain ways while under the influence of gravity. This leads to complex, non-linear interactions between the agent and the environment such as contacts between robot and object, object and table, and robot and table. Overall, this implies that to learn effective policies an agent must collect a large amount of relevant experience and also use it efficiently for learning.

5.1. Tasks

We consider a total of 6 tasks grouped into two parts: Four hard tasks and two easy tasks. Hard and easy is defined by the success reward achieved by off-policy (in particular, PQL) methods in these environments. In easy environments, even Q-learning-based off-policy methods can obtain non-zero performance but not in hard tasks. See appendix sec. A.

Hard Difficulty Tasks All four hard tasks are based on the `Allegro-Kuka` environments [24]. These consist of an Allegro Hand (16 DoF) mounted on a Kuka arm (7 dof). The performance of the agent in the above three tasks is measured by the successes metric which is defined as the number of successes in a single episode. Three tasks include:

- **Regrasping**: The object must be lifted from the table

and held near a goal position $\mathbf{g}_t \in \mathbb{R}^3$ for $K = 30$ steps. This is called a "success". The target position and object position are reset to a random location after every success.

- **Throw**: The object must be lifted from the table and thrown into a bucket at $\mathbf{g}_t \in \mathbb{R}^3$ placed out of reach of the arm. The bucket and the object position are reset randomly after every successful attempt.
- **Reorientation**: Pick up the object and reorient it to a particular target pose $\mathbf{g_t} \in \mathbb{R}^7$ (position + orientation). The target pose is reset once the agent succeeds. This means that the agents needs to reorient the object in different poses in succession, which may sometimes entail placing the objects on the table and lifting it up in a different way.
- **Two Arms Reorientation**: Similar to the reorientation task above, pick up the object and reorient it to a particular target pose. However, there are two arms in the system, adding the additional complexity of having to transfer objects between arms to reach poses in different regions of space.

Easy Difficulty Tasks: In addition, we test on the following dexterous hand tasks. As before, the observation space consists of the joint angles and velocities $\mathbf{q}_t, \dot{\mathbf{q}}_t$, object pose $\mathbf{x}_t$ and velocities $\mathbf{v}_t, \omega_t$.

- **Shadow Hand**: We test in-hand reorientation task of a cube using the 24-DoF Shadow Hand.
- **Allegro Hand**: This is the same as the previous in-hand reorientation task but with the 16-DoF Allegro Hand.

5.2. Baselines

We test against state-of-the-art RL methods designed for the GPU-accelerated large-scale setting we consider in this paper. We compare against both on-policy [24] and off-policy [15] variants as well as vanilla PPO [27].

- **PPO** (Proximal Policy Optimization) [27]: In our setting, we just increase the data throughput for PPO by increasing the batch size proportionately to the number of environments. In particular, we see over two orders of magnitude increase in the number of environments (from 128 to 24576).
- **Parallel Q-Learning** [15] A parallelized version of DDPG with different mixed exploration i.e. varying exploration noise across environments to further aid exploration. We use this baseline to compare if off-policy methods can outperform on-policy methods when the data collection capacity is high.
- **DexPBT** [24] A framework that combines population-based training with PPO. N Environments are divided into M groups, each containing $\frac{N}{M}$ environments. M separate policies are trained using PPO in each group of environments with different hyperparameters. At regular intervals, the worst-performing policies are replaced with the weights of best-performing policies and their hyperparameters are mutated randomly.

Due to the complexity of these tasks, experiments take about 48-60 hours on a single GPU, collecting $\approx$ 2e10 transitions. Since we run experiments on different machines, the wall clock time is not directly comparable and we compare runs against the number of samples collected. We run 5 seeds for each experiment and report the mean and standard error in the plots. In each plot, the solid line is $y(t) = \frac{1}{n}\sum_i y_i(t)$ while the width of the shaded region is determined by standard error $\frac{2}{\sqrt{n}}\sum_i (y(t) - y_i(t))^2$.

For each task, we use $M = 6$ policies for our method and DexPBT in a total of $N = 24576$ environments for each method. We use the dimension of learned parameter $\phi_j \in \mathbb{R}^{32}$ for the AllegroKuka tasks while we use $\phi_j \in \mathbb{R}^{16}$ for the ShadowHand and AllegroHand tasks since they are relatively simpler. We use a recurrent policy for the AllegroKuka tasks and an MLP policy for the Shadow Hand and Allegro Hand tasks and use PPO to train them. We collect 16 steps of experience per instance of the environment before every PPO update step. For SAPG, we tune the entropy coefficient σ by choosing the best from a small set $\{0, 0.003, 0.005\}$ for each environment. We find that $\sigma = 0$ works best for all AllegroHand, Regrasping, and Throw while $\sigma = 0.005$ works better for ShadowHand and Reorientation.

6. Results and Analysis

In the large-scale data setting, we are primarily concerned with optimality while sample-efficiency and wall-clock time are secondary concerns. This is because data is readily available—one only needs to spin up more GPUs, what is really important is how well our agent performs in the downstream tasks. Indeed, this aligns with how practitioners use RL algorithms in practice [1, 3, 23], where agents are trained with lots of domain randomization in large simulations and the primary concern is how well the agent can adapt and learn in these environments since this directly translates to real-world performance.

6.1. AllegroKuka tasks

The AllegroKuka tasks (Throw, Regrasping, Reorientation, Two Arms Reorientation) are hard due to large degrees of freedom. The environment also offers the possibility of many emergent strategies such as using the table to reorient the cube, or using gravity to reorient the cube. Therefore, a large amount of data is required to attain good performance on these tasks. Following Petrenko et al. [24] we use the number of successes as a performance metric on these tasks. Note that the DexPBT baseline directly optimizes for success by mutating the reward scales to achieve higher

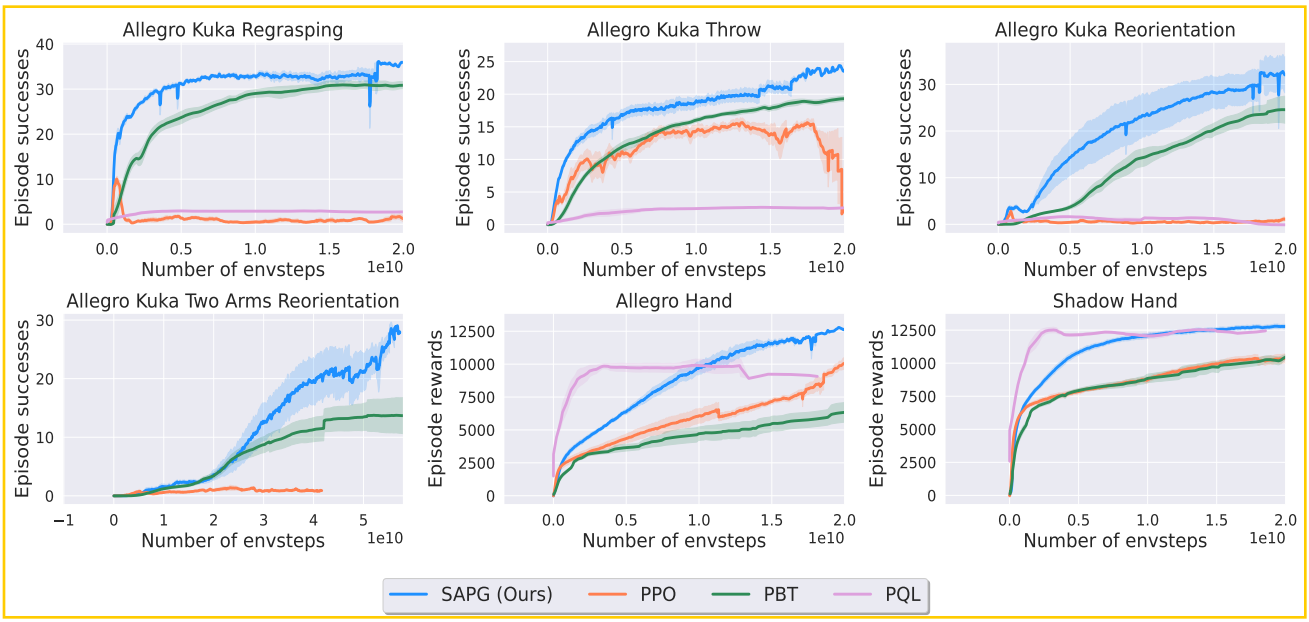

Figure 5. Performance curves of SAPG with respect to PPO, PBT and PQL baselines. On AllegroKuka tasks, PPO and PQL barely make progress and SAPG beats PBT. On Shadow Hand and Allegro Kuka Reorientatio and Two Arms Reorientation, SAPG performs best with an entropy coefficient of 0.005 while the coefficient is 0 for other environments. On ShadowHand and AllegroHand, while PQL is initially more sample efficient, SAPG is more performant in the longer run. AllegroKuka environments use successes as a performance metric while AllegroHand and ShadowHand use episode rewards.

Task	PPO [27]	PBT [24]	PQL [15]	SAPG ($\lambda_{\text{ENT}} = 0$)	SAPG ($\lambda_{\text{ENT}} = 0.005$)
AllegroHand	1.01e4 ± 6.31e2	7.28e3 ± 1.24e3	1.01e4 ± 5.28e2	**1.23e4 ± 3.29e2**	9.14e3 ± 8.38e2
ShadowHand	1.07e4 ± 4.90e2	1.01e4 ± 1.80e2	1.28e4 ± 1.25e2	1.17e4 ± 2.64e2	**1.28e4 ± 2.80e2**
Regrasping	1.25 ± 1.15	31.9 ± 2.26	2.73 ± 0.02	**35.7 ± 1.46**	33.4 ± 2.25
Throw	16.8 ± 0.48	19.2 ± 1.07	2.62 ± 0.08	**23.7 ± 0.74**	18.7 ± 0.43
Reorientation	2.85 ± 0.05	23.2 ± 4.86	1.66 ± 0.11	33.2 ± 4.20	**38.6 ± 0.63**
Two Arms Reorientation	1.73 ± 0.51	14.46 ± 2.91	-	-	**28.58 ± 1.55**

Table 1. Performance after 2e10 samples for different methods with standard error. This is measured by successes for the AllegroKuka tasks and by episode rewards for in-hand reorientation tasks. Across environments, we find that our method performs better than baselines.

success rate, whereas our method can only optimize a fixed reward function. Despite this, we see that SAPG achieves a $12 - 66\%$ higher success rate than DexPBT on regrasping, throw and reorientation. SAPG performs 66% better than PBT on the challenging reorientation task. SAPG fairs even better on the two-arm reorientation, obtaining more than twice the number of successes on average compared to PBT. Note that vanilla PPO and PQL are unable to learn any useful behaviors on these hard tasks.

6.2. In-hand reorientation

The AllegroHand and ShadowHand reorientation tasks from Li et al. [15] are comparatively easier since they have lower degrees of freedom and the object doesn't move around much and remains inside the hand. On these tasks, we observe that PQL and PPO are able to make significant progress. In particular, we find that PQL is very sample-efficient because it is off-policy and utilizes past data for updates. However, we find that SAPG achieves higher *asymptotic performance*. This is because on-policy methods are better at latching onto high reward trajectories and do not have to wait several iterations for the Bellman backup to propagate back to initial states. As discussed previously, in large-scale settings in simulation, we are primarily concerned with asymptotic performance since we want to maximize the downstream performance of our agents (within a reasonable training time budget). We see that on AllegroHand, SAPG beats PQL by a 21% margin, while on the ShadowHand task it achieves comparable performance. On these tasks, both PBT and PPO generally perform worse. This is because PPO is not able to efficiently leverage the large batch size. PBT loses the benefit of its hyperparameter

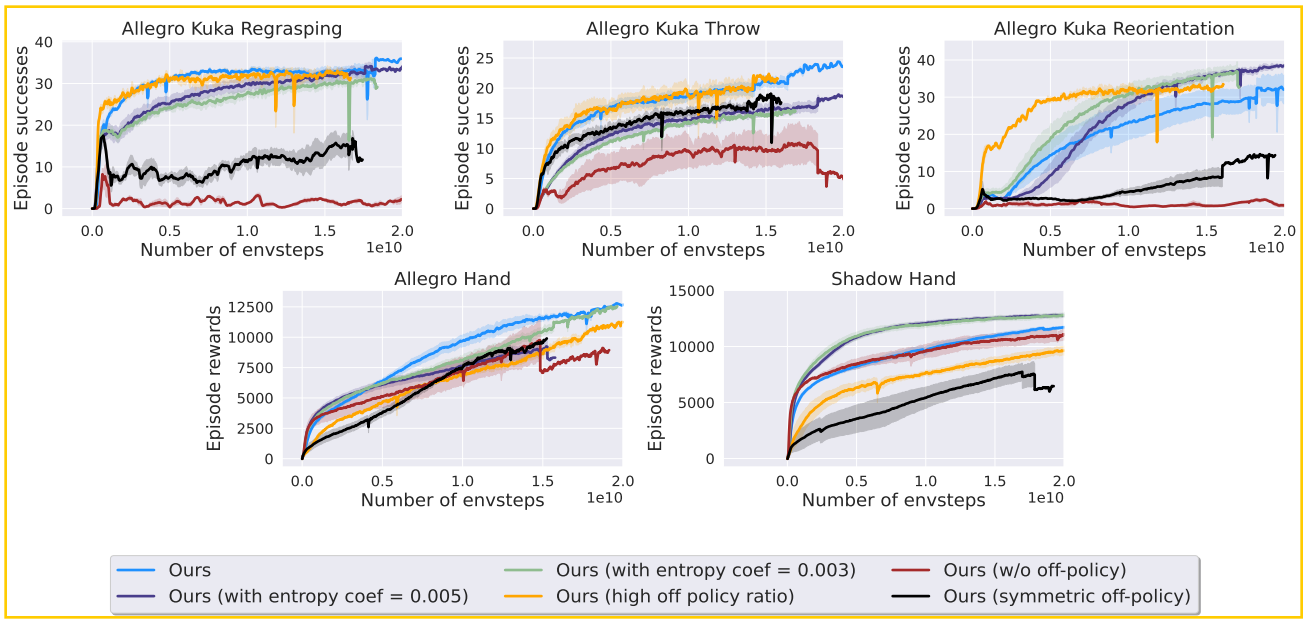

Figure 6. Performance curves for ablations of our method. The variants of our method with a symmetric aggregation scheme or without an off-policy combination perform significantly worse. Entropy regularization affects performance across environments, giving a benefit in reorientation. Using a high off-policy ratio without subsampling data leads to worse performance on ShadowHand and AllegroHand.

mutation because the environment is simpler and the default hyperparameters work well, so it roughly reduces to simple PPO in $\frac{N}{M}$ environments.

6.3. Ablations

The core idea behind SAPG is to combine data from different policies instead of optimizing a single policy with an extremely large batch. In this section, we will analyze our specific design choices for how we combine data (choice of i and $\mathcal{X}$ and λ) and for how we enforce diversity among the data collected by the policies. In particular, we have the following variants

- **SAPG (with entropy coef)** As discussed in sec. 5.2 , here we add an entropy loss to the followers to encourage data diversity. We explore different choices for the scaling coefficient of this loss $\sigma \in \{0, 0.005, 0.003\}$.
- **SAPG (high off-policy ratio)** In SAPG, when updating the leader, we subsample the off-policy data from the followers such that the off-policy dataset size matches the on-policy data. This is done because off-policy data is typically noisier and we do not want to drown out the gradient from on-policy data. In SAPG with a high off-policy ratio, we remove the subsampling step and instead see the impact of computing the gradient on the *entire* combined off-policy + on-policy dataset.
- **Ours (symmetric)** In SAPG, we choose $i = 1$ to be the "leader" and the rest are "followers". Only the leader receives off-policy data while the followers use the standard on-policy loss. A natural alternative is where there are no privileged policies and each policy is updated with off-policy data from all others as discussed in sec. 4.2.

We observe that SAPG outperforms or achieves comparable performance to the entropy-regularized variant except in the Reorientation environment where the variant with coefficient $5e-3$ performs up to 16.5% better. Reorientation is one of the harder tasks out of the four AllegroKuka tasks and has a lot of scope for learning emergent strategies such as using the table to move the object around, etc. Explicit exploration might be useful in discovering these behaviors.

The variant of ours which uses all the off-policy data is significantly worse on the AllegroHand and ShadowHand tasks and marginally worse on Regrasping and Throw environments. It is more sample efficient than SAPG on Reorientation but achieves lower asymptotic performance. This could be because in the simple environments, additional data has marginal utility. In the harder AllegroKuka environments, it is beneficial to use all the data initially since it may contain optimal trajectories that would otherwise be missed. However, once an appreciable level of performance is achieved, it becomes better to subsample to prevent the noise in the off-policy update from drowning out the on-policy gradient.

Finally, the symmetric variant of our method performs significantly worse across the board. This is possibly because using all the data to update each policy leads to them converging in behavior. If all the policies start executing the same actions, the benefit of data diversity is lost and SAPG

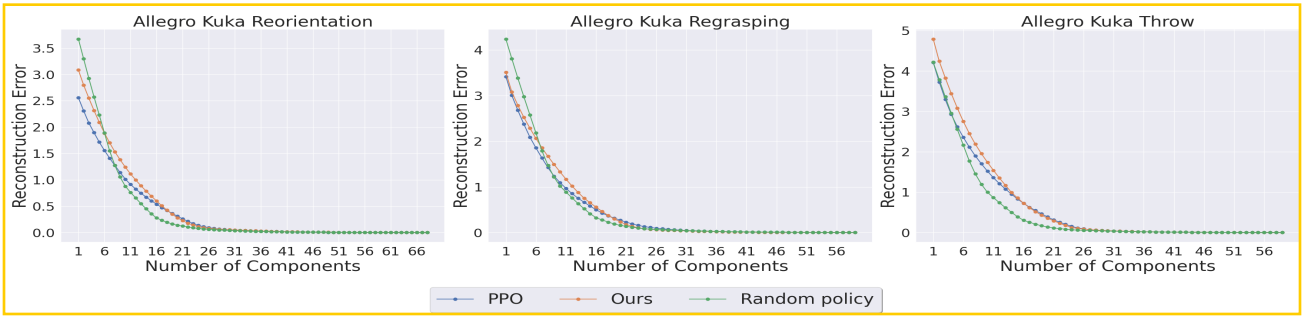

Figure 7. Curves comparing reconstruction error for states visited during training using top-k PCA components for SAPG (Ours), PPO and a randomly initialized policy

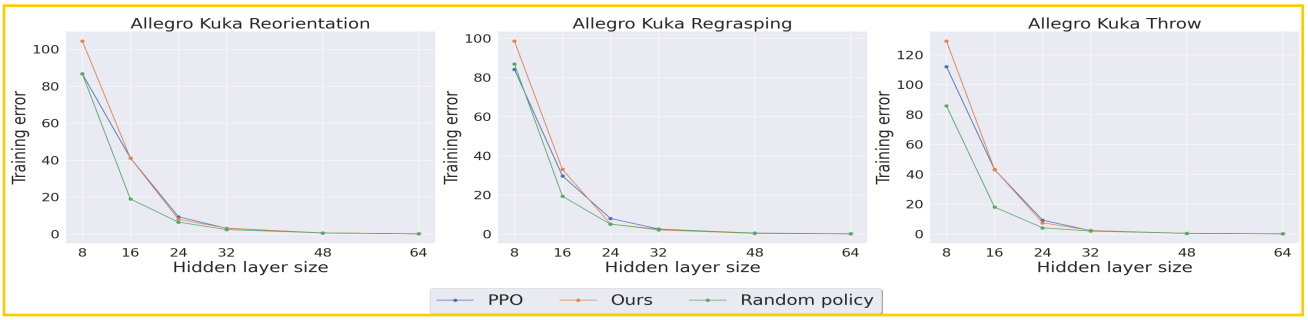

Figure 8. Curves comparing reconstruction error for states visited during training using MLPs with varying hidden layer dimensions for SAPG (Ours), PPO and a randomly initialized policy

reduces to vanilla PPO. Of course, there is a rich space of possible algorithms depending on particular choices of how data is aggregated and diversity is encouraged of which we have explored a small fraction.

6.4. Diversity in exploration

To analyze why our method outperforms the baseline method, we conduct experiments comparing the diversity of states visited by each algorithm during training. We devise two metrics to measure the diversity of the state space and find that our method beats PPO in both metrics.

- **PCA** - We compute the reconstruction error of a batch of states using k most significant components of PCA and plot this error as a function of k. In general, a set that has variation along fewer dimensions of space can be compressed with fewer principal vectors and will have lower reconstruction error. This metric therefore measures the extent to which the policy explores different dimensions of state space. Figure-7 contains the plots for this metric. We find that the rate of decrease in reconstruction error with an increase in components is the slowest for our method.
- **MLP** - We train feedforward networks with small hidden layers on the task of input reconstruction on batches of environment states visited by our algorithm and PPO during training. The idea behind this is that if a batch of states has a more diverse data distribution then it should be harder to reconstruct the distribution using small hidden layers because high diversity implies that the distribution is less compressible. Thus, high training error on a batch of states is a strong indicator of diversity in the batch. As can be observed from the plots in Figure-8, we find that training error is consistently higher for our method compared to PPO across different hidden layer sizes.

7. Conclusion

In this work, we present a method to scale reinforcement learning to utilize large simulation capacity. We show how current algorithms obtain diminishing returns if we perform naive scaling by batch size and do not use the increased volume of data efficiently. Our method achieves state-of-the-art performance on hard simulation benchmarks.

Acknowledgements

We thank Alex Li and Russell Mendonca for fruitful discussions regarding the method and insightful feedback. We would also like to thank Mihir Prabhudesai and Kevin Gmelin for proofreading an earlier draft. This project was supported in part by ONR N00014-22-1-2096 and NSF NRI

IIS-2024594.

References

[1] Ananye Agarwal, Ashish Kumar, Jitendra Malik, and Deepak Pathak. Legged locomotion in challenging terrains using egocentric vision. In *Conference on Robot Learning*, 2022. URL https://api.semanticscholar.org/CorpusID:252733339.

[2] Tao Chen, Jie Xu, and Pulkit Agrawal. A system for general in-hand object re-orientation, 2021.

[3] Xuxin Cheng, Kexin Shi, Ananye Agarwal, and Deepak Pathak. Extreme parkour with legged robots. *ArXiv*, abs/2309.14341, 2023. URL https://api.semanticscholar.org/CorpusID:262826068.

[4] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus), 2016.

[5] Thomas Degris, Martha White, and Richard S. Sutton. Off-policy actor-critic. *CoRR*, abs/1205.4839, 2012. URL http://arxiv.org/abs/1205.4839.

[6] Lasse Espeholt, Hubert Soyer, Rémi Munos, Karen Simonyan, Volodymyr Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. IMPALA: scalable distributed deep-rl with importance weighted actor-learner architectures. *CoRR*, abs/1802.01561, 2018. URL http://arxiv.org/abs/1802.01561.

[7] Rasool Fakoor, Pratik Chaudhari, and Alexander J. Smola. P3o: Policy-on policy-off policy optimization. In Ryan P. Adams and Vibhav Gogate, editors, *Proceedings of The 35th Uncertainty in Artificial Intelligence Conference*, volume 115 of *Proceedings of Machine Learning Research*, pages 1017–1027. PMLR, 22–25 Jul 2020. URL https://proceedings.mlr.press/v115/fakoor20a.html.

[8] Zipeng Fu, Xuxin Cheng, and Deepak Pathak. Deep whole-body control: Learning a unified policy for manipulation and locomotion. *ArXiv*, abs/2210.10044, 2022. URL https://api.semanticscholar.org/CorpusID:252968218.

[9] Tuomas Haarnoja, Aurick Zhou, P. Abbeel, and Sergey Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. *ArXiv*, abs/1801.01290, 2018. URL https://api.semanticscholar.org/CorpusID:28202810.

[10] Ankur Handa, Arthur Allshire, Viktor Makoviychuk, Aleksei Petrenko, Ritvik Singh, Jingzhou Liu, Denys Makoviichuk, Karl Van Wyk, Alexander Zhurkevich, Balakumar Sundaralingam, Yashraj S. Narang, Jean-Francois Lafleche, Dieter Fox, and Gavriel State. Dextreme: Transfer of agile in-hand manipulation from simulation to reality. *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5977–5984, 2022. URL https://api.semanticscholar.org/CorpusID:253107794.

[11] W. K. Hastings. Monte Carlo sampling methods using Markov chains and their applications. *Biometrika*, 57 (1):97–109, 04 1970. ISSN 0006-3444.

[12] Dan Horgan, John Quan, David Budden, Gabriel Barth-Maron, Matteo Hessel, Hado van Hasselt, and David Silver. Distributed prioritized experience replay. *CoRR*, abs/1803.00933, 2018. URL http://arxiv.org/abs/1803.00933.

[13] Sham Kakade and John Langford. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274, 2002.

[14] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf.

[15] Zechu Li, Tao Chen, Zhang-Wei Hong, Anurag Ajay, and Pulkit Agrawal. Parallel q-learning: Scaling off-policy reinforcement learning under massively parallel simulation, 2023.

[16] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning, 2019.

[17] Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, Miles Macklin, David Hoeller, Nikita Rudin, Arthur Allshire, Ankur Handa, et al. Isaac gym: High performance gpu-based physics simulation for robot learning. *arXiv preprint arXiv:2108.10470*, 2021.

[18] Wenjia Meng, Qian Zheng, Gang Pan, and Yilong Yin. Off-policy proximal policy optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37 (8):9162–9170, June 2023. ISSN 2159-5399. doi:

10.1609/aaai.v37i8.26099. URL http://dx.doi.org/10.1609/aaai.v37i8.26099.

[19] Takahiro Miki, Joonho Lee, Jemin Hwangbo, Lorenz Wellhausen, Vladlen Koltun, and Marco Hutter. Learning robust perceptive locomotion for quadrupedal robots in the wild. *Science Robotics*, 7(62):eabk2822, 2022.

[20] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *ArXiv*, abs/1312.5602, 2013. URL https://api.semanticscholar.org/CorpusID:15238391.

[21] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. *CoRR*, abs/1602.01783, 2016. URL http://arxiv.org/abs/1602.01783.

[22] 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. Massively parallel methods for deep reinforcement learning. *CoRR*, abs/1507.04296, 2015. URL http://arxiv.org/abs/1507.04296.

[23] OpenAI, Marcin Andrychowicz, Bowen Baker, Maciek Chociej, Rafal Józefowicz, Bob McGrew, Jakub Pachocki, Arthur Petron, Matthias Plappert, Glenn Powell, Alex Ray, Jonas Schneider, Szymon Sidor, Josh Tobin, Peter Welinder, Lilian Weng, and Wojciech Zaremba. Learning dexterous in-hand manipulation. *CoRR*, abs/1808.00177, 2018. URL http://arxiv.org/abs/1808.00177.

[24] Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training. In Kostas E. Bekris, Kris Hauser, Sylvia L. Herbert, and Jingjin Yu, editors, *Robotics: Science and Systems XIX, Daegu, Republic of Korea, July 10-14, 2023*, 2023. doi: 10.15607/RSS.2023.XIX.037. URL https://doi.org/10.15607/RSS.2023.XIX.037.

[25] Nikita Rudin, David Hoeller, Philipp Reist, and Marco Hutter. Learning to walk in minutes using massively parallel deep reinforcement learning, 2022.

[26] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In Francis Bach and David Blei, editors, *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pages 1889–1897, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schulman15.html.

[27] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

[28] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018.

[29] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. In S. Solla, T. Leen, and K. Müller, editors, *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

[30] Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H. Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019. URL https://api.semanticscholar.org/CorpusID:204972004.

[31] Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Rémi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *CoRR*, abs/1611.01224, 2016. URL http://arxiv.org/abs/1611.01224.

[32] Christopher Watkins and Peter Dayan. Q-learning. *Machine Learning*, 8:279–292, 1992. URL https://api.semanticscholar.org/CorpusID:208910339.

[33] Ronald J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 2004. URL https://api.semanticscholar.org/CorpusID:19115634.

A. Task and Environment Details

Hard Difficulty Tasks All four hard tasks are based on the `Allegro-Kuka` environments[24]. These consist of an Allegro Hand (16 dof) mounted on a Kuka arm (7 dof). In each case, the robot must manipulate a cuboidal kept on a fixed table. The observation space is $\mathbf{o}_t = [\mathbf{q}, \dot{\mathbf{q}}, \mathbf{x}_t, \mathbf{v}_t, \omega_t, \mathbf{g}_t, \mathbf{z}_t]$, where $\mathbf{q}, \dot{\mathbf{q}} \in \mathbb{R}^{23}$ are the joint angles and velocities respectively of each joint of the robot, $\mathbf{x}_t \in \mathbb{R}^7$ is the pose of the object, $\mathbf{v}_t$ is its linear velocity and ω_t is its angular velocity, $\mathbf{g}_t$ is a task-dependent goal observation and $\mathbf{z}_t$ is auxiliary information pertinent to solving the task such as if the object has been lifted. These tasks consist of a complex environment but a simple reward function allowing opportunities for emergent strategies to be learnt such as in-hand reorientation under the influence of gravity, reorientation against the table, different types of throws and grasps and so on. The performance of the agent in the above three tasks is measured by the successes metric which is defined as the number of successes in a single episode. Three tasks include:

- **Regrasping** - The object must be lifted from the table and held near a goal position $\mathbf{g}_t \in \mathbb{R}^3$ for $K = 30$ steps. This is called a "success". The target position and object position are reset to a random location after every success. The success tolerance δ defines the maximum error between object pose and goal pose for a success $\|\mathbf{g}_t - (\mathbf{x}_t)_{0:3}\| < \delta$. This tolerance is decreased in a curriculum from 7.5cm to 1cm, decremented by 10% each time the the average number of successes in an episode crosses 3. The reward function is a weighted combination of rewards encouraging the hand to reach the object r_{reach}, a bonus r_{lift}, rewards encouraging the hand to move to goal location after lifting r_{target} and a success bonus $r_{success}$.
- **Throw** - The object must be lifted from the table and thrown into a bucket at $\mathbf{g}_t \in \mathbb{R}^3$ placed out of reach of the arm. The bucket and the object position are reset randomly after every successful attempt. The reward function is similar to regrasping with the difference being that the target is now a bucket instead of a point.
- **Reorientation** - This task involves picking up the object and reorienting it to a particular target pose $\mathbf{g}_t \in \mathbb{R}^7$ (position + orientation). Similar to the regrasping task, there is a success tolerance δ which is varied in a curriculum. The target pose is reset once the agent succeeds. This means the agents needs to the object in different poses in succession, which may sometimes entail placing the obxfject on the table and lifting it up in a different way. Here too, the reward function is similar to regrasping, with the goal now being a pose in $\mathbb{R}^7$ instead of $\mathbb{R}^3$.
- **Two Arms Reorientation**: The objective is the same to the AllegroKuka Reorientation task. The difference is that the setup has another arm now. In this task, the target pose may be within the reach of one arm but not the other, which means that the arms may need to transfer the object between themselves, for which it needs to learn complex throwing and catching behaviours.

Easy Difficulty Tasks: In addition, we test on the following dexterous hand tasks. As before, the observation space consists of the joint angles and velocities $\mathbf{q}_t, \dot{\mathbf{q}}_t$, object pose $\mathbf{x}_t$ and velocities $\mathbf{v}_t, \omega_t$. Following previous works [15], we use the net episode reward as a performance metric for the ShadowHand and AllegroHand tasks.

- **Shadow Hand**: We test on in-hand reorientation task of a cube using the 24-DoF Shadow Hand([23]). The task is to attain a specified goal orientation (specified as a quaternion) for the cube $\mathbf{g}_t \in \mathbb{R}^4$. The reward is a combination of the orientation error and a success bonus.
- **Allegro Hand**: This is the same as the previous in-hand reorientation task but with the 16-DoF Allegro Hand instead.

B. Training hyperparameters

We use two different sets of default hyperparaeters for PPO in AllegroKuka and Shadow Hand tasks which are descibed below.

B.1. AllegroKuka tasks

We use a Gaussian policy where the mean network is an LSTM with 1 layer containing 768 hidden units. The observation is also passed through an MLP of with hidden layer dimensions $768 \times 512 \times 256$ and an ELU activation [4] before being input to the LSTM. The sigma for the Gaussian is a fixed learnable vector independent of input observation.

Hyperparameter	Value
Discount factor, γ	0.99
τ	0.95
Learning rate	1e-4
KL threshold for LR update	0.016
Grad norm	1.0
Entropy coefficient	0
Clipping factor ϵ	0.1
Mini-batch size	num_envs $\cdot$ 4
Critic coefficient λ'	4.0
Horizon length	16
LSTM Sequence length	16
Bounds loss coefficient	0.0001
Mini epochs	2

Table 2. Training hyperparameters for AllegroKuka tasks

B.2. Shadow Hand

We use a Gaussian policy where the mean network is an MLP with hidden layers dimensions $512 \times 512 \times 256 \times 128$ and an ELU activation [4]

Hyperparameter	Value
Discount factor, γ	0.99
τ	0.95
Learning rate	5e-4
KL threshold for LR update	0.016
Grad norm	1.0
Entropy coefficient	0
Clipping factor ϵ	0.1
Mini-batch size	num_envs $\cdot$ 4
Critic coefficient λ'	4.0
Horizon length	8
Bounds loss coefficient	0.0001
Mini epochs	5

Table 3. Training hyperparameters for Shadow Hand

B.3. Allegro Hand

We use a Gaussian policy where the mean network is an MLP with hidden layers dimensions $512 \times 256 \times 128$ and an ELU activation.

Hyperparameter	Value
Discount factor, γ	0.99
τ	0.95
Learning rate	5e-4
KL threshold for LR update	0.016
Grad norm	1.0
Entropy coefficient	0
Clipping factor ϵ	0.2
Mini-batch size	num_envs $\cdot$ 4
Critic coefficient λ'	4.0
Horizon length	8
Bounds loss coefficient	0.0001
Mini epochs	5

Table 4. Training hyperparameters for Shadow Hand

Note: In case of experiments with entropy based exploration, each block of environments has it's own learnable vector sigma which enable policies for different blocks to have different entropies.