ROLLOUT AGGREGATION STABILIZES ON-POLICY REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale parallel PPO can struggle on complex, long-horizon tasks with high-dimensional action spaces. Prior work has emphasized exploration bottlenecks and diminishing returns from increasing environment parallelism. We instead present evidence that learning instability offers a compelling alternative explanation for PPO's poor performance. To improve stability, we introduce rollout aggregation (RAGGER), a minimal modification that retains the previous rollout and trains on it alongside newly collected samples. We show that overlapping rollout batches regularize the policy's optimization trajectory by preserving pressure to retain recently learned behavior while incorporating new experience. Across several challenging manipulation tasks, RAGGER prevents the collapse observed with standard PPO and outperforms strong baselines, including SAPG and CPO, with several-fold improvements in sample efficiency. Our findings motivate studying recent-data reuse as a mechanism for stabilizing policy learning and achieving strong, sample-efficient performance at scale.

1 INTRODUCTION

On-policy RL, and PPO (Schulman et al., 2017b) in particular, has enabled control of high DoF contact-rich systems such as humanoids and dexterous hands by learning from tens of thousands of parallel simulation environments (Makoviychuk et al., 2021). Unfortunately, for many very long-horizon dexterous manipulation tasks, Singla et al. (2024) show that simply scaling the number of environments has diminishing returns, and PPO stops improving as parallelism grows. The prevailing explanation is limited exploration: Singla et al. (2024) argue that a single Gaussian policy concentrates its samples near its mean, so additional environments mostly produce duplicate data. Prior work therefore scales PPO with population-based training (Petrenko et al., 2023) or trains multiple policies to diversify the data (Singla et al., 2024; Shitanda et al., 2026).

We bring an alternate perspective: on several of these tasks, the bottleneck is *stability, not diversity*. PPO often does discover good behavior, but then loses it; performance rises and collapses rather than plateauing (Sec. 4.1). This view is consistent with recent evidence that PPO's plateaus at scale need not stem from exploration (Beukman et al., 2026).

To address this instability, we minimally modify PPO: each update trains on the newly collected rollout together with the one collected immediately before it. We call this **rollout aggregation** (RAGGER). Overlapping consecutive batches preserve pressure to retain recently learned behavior while still incorporating new experience. This single change prevents PPO's collapse and outperforms SAPG (Singla et al., 2024) and CPO (Shitanda et al., 2026) on several challenging long-horizon dexterous manipulation tasks (Petrenko et al., 2023), matching their final performance with up to 4× fewer samples. It also improves over PPO on several object-tracking tasks (Mandi et al., 2025), another class of long-horizon, contact-rich manipulation problems.

2 BACKGROUND

2.1 POLICY GRADIENT METHODS

We consider the usual reinforcement learning setting where each task is modeled as an infinite-horizon discounted MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

$\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$[1], reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, initial-state distribution ρ, and discount factor $\gamma \in [0, 1)$. The objective is to find a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the expected cumulative discounted return from $t = 1$ to ∞: $\pi^* = \arg\max_\pi J(\pi)$ with

$$J(\pi) = \mathbb{E}_{s_1 \sim \rho, a_t \sim \pi(s_t), s_{t+1} \sim \mathcal{P}(s_t, a_t)} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \right]. \tag{1}$$

Given a policy π, we define the state-value function $V^\pi(s) := \mathbb{E}_\pi[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) | s_1 = s]$ and action-value function $Q^\pi(s, a) := \mathbb{E}_\pi[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) | s_1 = s, a_1 = a]$. The *advantage* $A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$ measures how much better the action a is than the policy's average action at that state s.

Policy gradient methods optimize $J(\pi)$ by iteratively estimating the gradient of the objective with respect to the policy parameters θ and performing gradient ascent; we refer the reader to Section A.1 for more details. In practice, one uses a finite batch of trajectories $\mathcal{D}$ to estimate the expectation and gradient: Proximal Policy Optimization (Schulman et al., 2017b) proposes a clipped surrogate RL objective that forces small, stable policy updates. Using $\rho_\theta := \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$, the PPO objective is

$$\hat{J}^{\text{PPO}}(\pi_{\text{old}}, \pi_\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,a) \in \mathcal{D}} \min \left(\rho_\theta \hat{A}^{\pi_{\theta_{\text{old}}}}(s, a), \text{clip}\left(\rho_\theta, 1 - \epsilon, 1 + \epsilon\right) \hat{A}^{\pi_{\theta_{\text{old}}}}(s, a) \right). \tag{2}$$

Here ϵ is a clipping hyperparameter. The empirical advantage $\hat{A}^{\pi_{\theta_{\text{old}}}}(s, a)$ is typically computed using generalized advantage estimation (GAE) (Schulman et al., 2018).

PPO has been shown to be a simple and effective policy gradient method that is widely used in practice, especially in large-scale RL training (Makoviychuk et al., 2021).

2.2 Large-scale Parallel PPO

In recent years, GPU-based simulators (Liang et al., 2018; Makoviychuk et al., 2021; NVIDIA et al., 2025; Zakka et al., 2026; Genesis, 2026) have enabled large-scale parallel RL training. Thousands of environments are simulated in parallel allowing for rapid rollout data collection. This has allowed techniques like PPO to scale to challenging tasks. Each training epoch consists of two steps.

Rollout. $N > 10^3$ environments are rolled out in parallel, each running the same policy π_θ to collect a batch of trajectories $\mathcal{D}$ (Schulman et al., 2017b). Note that in practice, the rollout does not necessarily require a complete episode or hundreds of timesteps per environment (Andrychowicz et al., 2020; Makoviychuk et al., 2021); one rolls out each environment for a constant buffer horizon H_{buf} timesteps, often as small as 16 or 32 frames (Makoviychuk et al., 2021; Mayor et al., 2025; Beukman et al., 2026). Any remaining return from the end states of the buffer is estimated using value bootstrapping (Pardo et al., 2018). Once every (s, a) pair in the rolled out batch has an estimated value and advantage and has been aggregated, the rollout step concludes.

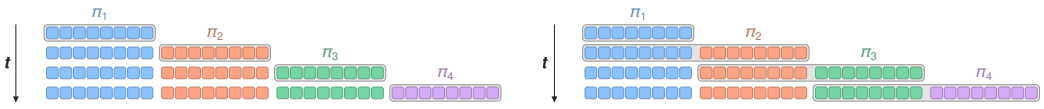
Optimization. The collected rollout dataset is split into a few minibatches; the policy is updated after each minibatch in accordance with the PPO objective (Equation (2)). Using every minibatch once constitutes a mini-epoch, and a few mini-epochs are repeated to conclude one epoch. Following this we go back to the rollout step with a new policy (Schulman et al., 2017b; Makoviichuk & Makoviychuk, 2021; Makoviychuk et al., 2021).

Recent works improving PPO modify either the rollout or optimization setup in various ways to encourage more stable or sample-efficient learning (Singla et al., 2024; Shitanda et al., 2026; Beukman et al., 2026). See Section 5 for a discussion of these methods.

3 Rollout Aggregation

We propose a simple minimal modification to PPO to improve its stability, and in all our evaluations this modification is sufficient to stably and monotonically improve performance on challenging long-horizon tasks. The key idea is very simple: instead of discarding the previous rollout batch after

[1]Here, $\Delta(X)$ denotes the set of all probability distributions over set X.



(a) Buffer use in PPO

(b) Buffer use in RAGGER

Figure 1: An illustration of our method. (a) Standard PPO trains only on the current rollout batch. (b) RAGGER trains on the current rollout batch together with state-action pairs from the previous rollout batch. This substantially stabilizes learning and prevents forgetting of recent behavior (see Section 4.4).

each update, we retain it and train on it alongside the freshly collected samples. We call this **rollout aggregation** (RAGGER). Figure 1 illustrates the method. We now make this precise (an algorithm is presented in Algorithm 1).

Suppose that for each $k \geq 1$, the policy rolling out data for epoch k is $\pi_k \equiv \pi_{\theta_k}$, with initial policy π_1. Let the value function or critic prior to epoch k be denoted $\hat{V}_k \equiv \hat{V}_{\phi_k}$. Let $\mathcal{D}_k$ denote the rollout batch collected by π_k for epoch k by stepping N environments for H_{buf} timesteps so $|\mathcal{D}_k| = NH_{\text{buf}} \equiv D$.

We train both the policy and critic on the aggregated dataset $\mathcal{D} \equiv \mathcal{D}_{k-1} \cup \mathcal{D}_k$. Note that every trajectory in $\mathcal{D}_{k-1}$ is continued by a corresponding trajectory in $\mathcal{D}_k$, so we may treat the two datasets as a single dataset of trajectories of length $2H_{\text{buf}}$ when computing the generalized advantage estimates. All value estimates in this computation use the current critic $\hat{V}_k$; in particular, the old critic $\hat{V}_{k-1}$ is not used. We note that strictly speaking, we mix actions from π_{k-1} and π_k with no off-policy correction, but this does not seem to be crucial (see Section 4.3). Call the resulting estimates $\hat{A}^{\mathcal{D}}(s,a)$. Define policy ratio $\rho^{(k)}(a|s)$ between the current policy π_θ and the policy π_k before epoch k as the importance sampling ratio between the two policies:

$$\rho^{(k)}(a|s) := \frac{\pi_\theta(a|s)}{\pi_k(a|s)}. \tag{3}$$

Our surrogate objective for epoch k is then the average of the two surrogate objectives for the two policies:

$$J^{\text{actor}}_{\text{RAGGER}}(\theta) = \frac{1}{2D}\left(J^{k,\pi_{k-1}}_{\text{RAGGER}}(\theta) + J^{k,\pi_k}_{\text{RAGGER}}(\theta)\right), \tag{4}$$

where (using the shorthand $\rho^{(\cdot)} := \rho^{(\cdot)}(a|s)$ and $\hat{A} := \hat{A}^{\mathcal{D}}(s,a)$)

$$J^{k,\pi_{k-1}}_{\text{RAGGER}}(\theta) = \sum_{(s,a)\in\mathcal{D}_{k-1}} \min\left(\rho^{(k-1)}\hat{A},\ \text{clip}(\rho^{(k-1)}, 1-\epsilon, 1+\epsilon)\hat{A}\right), \tag{5}$$

$$J^{k,\pi_k}_{\text{RAGGER}}(\theta) = \sum_{(s,a)\in\mathcal{D}_k} \min\left(\rho^{(k)}\hat{A},\ \text{clip}(\rho^{(k)}, 1-\epsilon, 1+\epsilon)\hat{A}\right). \tag{6}$$

Note that the importance sampling ratios for each (s,a) are computed with respect to the policy that collected it in the same spirit as in the TRPO surrogate objective. The value function $\hat{V}_\phi(s)$ is also similarly trained on the aggregated dataset $\mathcal{D}_{k-1} \cup \mathcal{D}_k$ using the same generalized advantage estimates $\hat{A}^{\mathcal{D}}(s,a)$ and the usual PPO critic loss:

$$\hat{L}^{\text{critic}}_{\text{RAGGER}}(\phi) = \frac{1}{2D} \sum_{(s,a)\in\mathcal{D}_{k-1}\cup\mathcal{D}_k} \left(\hat{V}_\phi(s) - (\hat{V}_k(s) + \hat{A})\right)^2. \tag{7}$$

4 Experiments and Analysis

We evaluate RAGGER on long-horizon dexterous manipulation (Sec. 4.1) and object tracking (Sec. 4.2), and ablate its design choices in Sec. 4.3.

Algorithm 1 The PPO+RAGGER algorithm. Lines marked ⋆ are the only deviations from PPO.

Input: initial policy π_1 and critic $\hat{V}_1$, N environments, rollout length H_{buf}, mini-epochs E, mini-batch size B, critic weight c_V, KL target d_{target}

$\mathcal{D}_0 \leftarrow \emptyset$; reset all environments
for epoch $k = 1, 2, \ldots$ **do**
  $\mathcal{D}_k \leftarrow \textsc{Rollout}(\pi_k, H_{\text{buf}})$
  $\mathcal{D} \leftarrow \mathcal{D}_{k-1} \cup \mathcal{D}_k$ as trajectories of length $2H_{\text{buf}}$ ⋆ $\mathcal{D}_{k-1}$ is carried over
  $\hat{A} \leftarrow \text{GAE}(\mathcal{D}, \hat{V}_k, \gamma, \lambda)$ ⋆ $2H_{\text{buf}}$ steps
  $(\theta, \phi) \leftarrow (\theta_k, \phi_k)$
  for $e = 1, \ldots, E$ **do**
    for minibatch $\mathcal{M} \subset \mathcal{D}$ of size B **do**
      $\rho \leftarrow \rho^{(k-1)}$ on $\mathcal{D}_{k-1}$, $\rho^{(k)}$ on $\mathcal{D}_k$ ⋆ IS ratios
      $(\theta, \phi) \leftarrow \textsc{Adam}(\nabla\big[- J^{\text{actor}}_{\text{RAGGER}}(\theta) + c_V \hat{L}^{\text{critic}}_{\text{RAGGER}}(\phi)\big]_{\mathcal{M}})$
  $(\theta_{k+1}, \phi_{k+1}) \leftarrow (\theta, \phi)$; $\mathcal{D}_{k-1} \leftarrow \mathcal{D}_k$ ⋆ fresh becomes carried

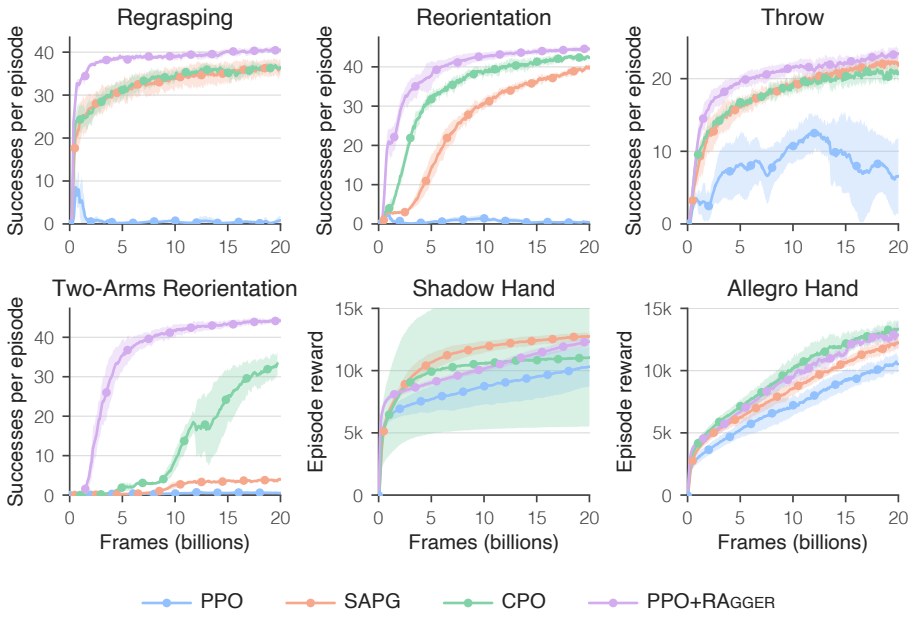


Figure 2: Training curves on six dexterous manipulation tasks: the four AllegroKuka tasks (Regrasping, Reorientation, Throw, Two-Arms Reorientation), Shadow Hand, and Allegro Hand. RAGGER learns fastest on all AllegroKuka tasks and is comparable to the best baseline on the Shadow Hand and Allegro Hand tasks. Lines show mean over 5 seeds, and shaded regions show ± 1 standard deviation.

4.1 ALLEGROKUKA TASKS

Environments. Following SAPG (Singla et al., 2024) and CPO (Shitanda et al., 2026), we evaluate on six dexterous manipulation tasks in Isaac Gym (Figure 6 in Section B.3). Four are the AllegroKuka tasks from DexPBT (Petrenko et al., 2023), in which a 16-DoF Allegro hand mounted on a 7 DoF Kuka arm must pick up an object and hold it at a target position (`Regrasping`), throw it into a container (`Throw`), or move it to a target position and orientation (`Reorientation`). The `Two-Arms Reorientation` task uses two arms and hands (for a total of 46 DoF), and requires passing the object from one hand to the other in addition to reorientation. Each episode is a sequence of attempts, and after every success the target is reset. These tasks therefore require long stretches of stable control.

Table 1: Performance after 2×10^{10} environment steps (24,576 environments): mean $\pm$ standard deviation over 5 seeds. Bold marks the best mean on each task. PPO+RAGGER outperforms all methods on the AllegroKuka tasks. On the two in-hand reorientation tasks, it is within 4% of the best method.

Task	PPO	SAPG	CPO	PPO+RAGGER
In-hand reorientation (episode reward ↑)				
Shadow Hand	10314 ±1589	**12752** ±308	11062 ±5537	12321 ±422
Allegro Hand	10515 ±700	12206 ±533	**13366** ±617	12891 ±1063
AllegroKuka (successes per episode ↑)				
Regrasping	0.85 ±1.15	36.40 ±2.12	36.36 ±1.00	**40.49** ±0.42
Reorientation	0.43 ±0.27	39.56 ±1.01	42.28 ±0.53	**44.58** ±0.63
Throw	6.56 ±5.22	21.82 ±1.09	20.79 ±1.35	**23.47** ±0.75
Two-Arms Reorientation	0.51 ±0.07	3.98 ±0.45	32.92 ±2.87	**44.13** ±0.61

The other two tasks, Shadow Hand and Allegro Hand (Makoviychuk et al., 2021) require rotating a cube in the hand to a target orientation.

As in prior work, we report the number of successes per episode for the AllegroKuka tasks, and the episode reward for the in-hand reorientation tasks.

Baselines. We compare against PPO, SAPG and CPO. SAPG and CPO are "leader-follower" methods designed for massively parallel training.

- **PPO**: Following SAPG, we run 24,576 environments and scale the minibatch size in proportion to the number of environments, which keeps the number of gradient steps per epoch fixed.
- **SAPG** (Split and Aggregate Policy Gradients): SAPG splits the environments evenly among six policies, one leader and five followers, which share a network backbone but are each conditioned on their own learned parameters. Each follower is trained with PPO on its own data, while the leader is trained on its own data plus a subsample of the followers' data, corrected with importance sampling.
- **CPO** (Coupled Policy Optimization): CPO argues that SAPG's followers drift too far from the leader, so the importance ratios on their data deviate strongly. It therefore keeps the followers close to the leader with a KL constraint, implemented as an advantage-weighted regression loss on the leader's data.

We omit DexPBT (Petrenko et al., 2023), which SAPG and CPO both report outperforming. Since RAGGER changes only PPO's update, it is also compatible with these methods; for instance, it could replace PPO inside population-based training.

Implementation. All methods use 24,576 parallel environments and are trained for 20B frames with 5 seeds. We implement RAGGER on top of the PPO implementation in `rl_games` (Makoviichuk & Makoviychuk, 2021), which we also use for the PPO baseline; for SAPG and CPO we use the official implementations. For RAGGER, the carried buffer is the same size as the rollout buffer for base PPO, so each update trains on the current rollout and the previous one. App. B.2 lists the hyperparameters for all methods.

4.2 HAND-OBJECT TRACKING TASKS

Robot learning via tracking retargeted 4D human hand-object interactions has recently emerged as a promising approach to learning dexterous manipulation skills (Mandi et al., 2025; Zhu et al., 2026), where a tracking reward encourages the agent to match the demonstrated hand and object poses frame by frame. Such tasks are naturally (i) high-dimensional and (ii) long-horizon, serving as a challenging testbed for RL.

Dexmachina (Mandi et al., 2025) presents a framework for bimanual manipulation of articulated household objects using *virtual object controllers* (VOC). They add a virtual PD controller in simulation to track the object's pose and articulation across the trajectory for the agent, and anneal

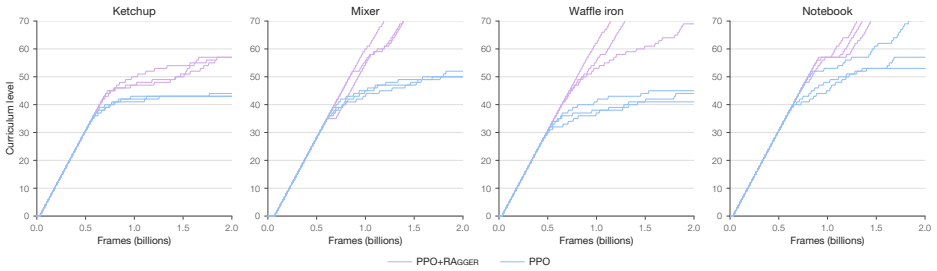
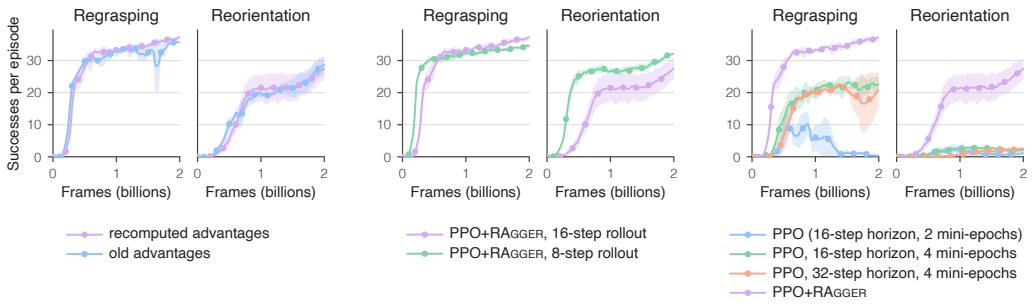


Figure 3: Training curves on four dexterous hand-object tracking tasks from Dexmachina (Mandi et al., 2025). RAGGER improves the performance of PPO on all four tasks. Lines show mean over 3 seeds, and shaded regions show ± 1 standard deviation.



(a) Frozen vs. recomputed advantages

(b) Same data per update as PPO

(c) Same passes per sample as RAGGER

Figure 4: What matters in RAGGER, on `Regrasping` and `Reorientation`. (a) Freezing each sample's advantage at collection time performs comparably to recomputing it, suggesting the gain comes from optimizing the previous rollout's surrogate. (b) Halving the rollout length so that each update uses as much data as PPO performs on par with the default. (c) Giving PPO as many passes per sample as RAGGER does not close the gap.

the strength (K_p) of this controller down to zero with time in a 70-step curriculum. We evaluate RAGGER on four of their harder tasks: MIXER300, NOTEBOOK300, KETCHUP300 and WAFFLE-IRON300 using the Inspire hand. Each task tracks a 300-frame human demonstration from ARCTIC (Fan et al., 2023). We change only the algorithmic component to RAGGER (Algorithm 1) and leave the rest of their framework, including the curriculum and all hyperparameters unchanged. Figure 3 shows that RAGGER improves on PPO on all four tasks.

We note that the improvement from using RAGGER is not primarily because of faster learning or sample efficiency: at equal frames, PPO matches it through the early part of training and only falls behind once PPO's progress stalls. We therefore attribute the gain to RAGGER continuing to make stable progress where PPO stagnates instead. We discuss this further in Section 4.4.

4.3 WHAT MATTERS IN ROLLOUT AGGREGATION?

We ablate a series of design choices to probe what matters for RAGGER. For the smaller-scale experiments, we use the single-arm `Reorientation` and `Regrasping` tasks. We use 3 seeds and run 2 billion frames for all methods.

Q: Does recomputing the advantages matter?

A: By default, RAGGER recomputes the advantages of the carried samples at every epoch (Sec. 3, Eq. 7): samples in $\mathcal{D}_{k-1}$ are re-evaluated with the current critic $\hat{V}_k$, and because each trajectory in $\mathcal{D}_{k-1}$ continues into $\mathcal{D}_k$, their GAE estimates extend over $2H_{\text{buf}}$ steps instead of H_{buf}. In this ablation, we instead freeze each sample's advantage at the value computed when it was collected,

i.e., from $\mathcal{D}_{k-1}$ alone with $\hat{V}_{k-1}$, and train the critic only on the fresh samples in $\mathcal{D}_k$. The algorithm then reduces to base PPO, except that the policy also optimizes the previous epoch's surrogate objective, unchanged (Eq. 4). This variant keeps only the policy-side aggregation, so it tests whether RAGGER's benefit comes from re-optimizing the previous surrogate alone, or also from re-evaluating the carried samples with the current critic.

Freezing the advantages performs comparably to recomputing them on both tasks (Fig. 4(a)). This suggests that RAGGER's benefit comes mainly from the policy re-optimizing its previous surrogate objective, while recomputing the advantages and training the critic on the carried samples adds little.

Q: Does RAGGER work simply because it trains on more data per update?

A: Because RAGGER trains on both $\mathcal{D}_{k-1}$ and $\mathcal{D}_k$, each update uses twice as much data as base PPO. To test whether this larger batch explains the gains, we halve the rollout length from $H_{\text{buf}} = 16$ to 8, so that the aggregated buffer ($8 + 8$ steps) is the same size as base PPO's (16 steps).

Halving the rollout length performs on par with the default, and slightly better on `Reorientation` (Fig. 4(b)). Since this variant trains on exactly as much data per update as base PPO, the larger batch does not explain RAGGER's gains. It does, however, still take twice as many gradient steps per collected sample as PPO, because every sample appears in two updates; we control for this next.

Q: Does RAGGER work simply because it reuses each sample more?

A: More optimization steps alone can improve PPO at this scale (Beukman et al., 2026), so we test whether the extra passes explain RAGGER's gains. We double PPO's number of miniepochs from 2 to 4, so that each sample receives as many passes as in RAGGER, and evaluate two rollout lengths: $H_{\text{buf}} = 16$, which matches RAGGER's rollout length, and $H_{\text{buf}} = 32$, which matches its data per update. Both variants also take as many gradient steps per collected frame as RAGGER. The remaining difference is *when* the passes happen: RAGGER splits each sample's passes across two consecutive updates, together with fresh data from the updated policy, whereas PPO performs all of them in a single update.

Neither variant closes the gap (Fig. 4(c)). Both improve over base PPO on `Regrasping` and perform similarly to each other, but both fail to take off on `Reorientation`, and on both tasks they remain well below RAGGER. Reusing each sample more, with either the update frequency or the data per update matched, therefore does not explain RAGGER's gains.

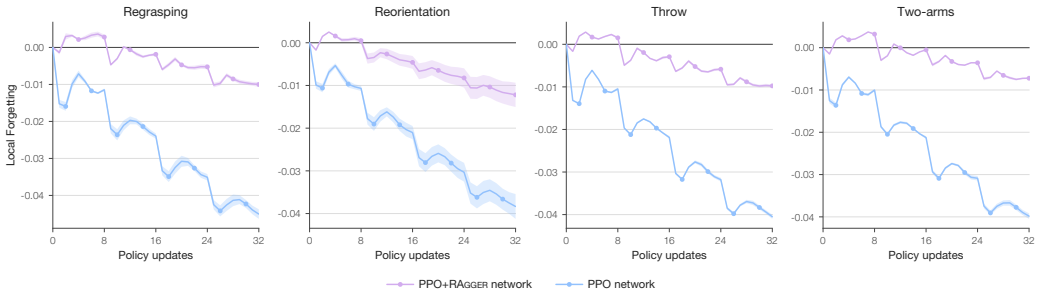
4.4 Local Forgetting Phenomena

To understand why RAGGER improves PPO stability, we introduce and examine the phenomenon of *local forgetting*. On-policy methods like PPO can "forget" good behavior learned in recently concluded rollouts, since the current policy is trained only on the current rollout and has no intrinsic incentive to preserve the objective on previous rollouts. It would be misguided to preserve the objective on much older rollouts, since the old estimated returns are stale and the policy has changed significantly; however, the few rollouts immediately preceding the current one are still relatively fresh, and the policy should be able to preserve its performance on them. We call the loss of performance on these recent rollouts *local forgetting*.

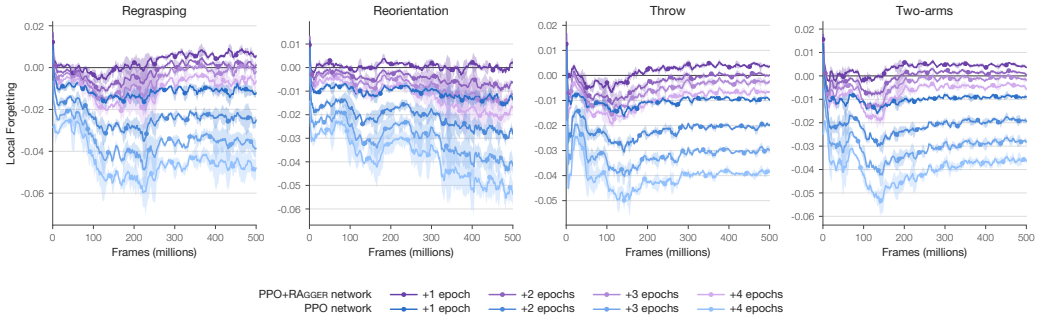
To test this, we perform the following training experiment. Each experiment consists of a series of *superepochs*, each superepoch consisting five epochs of training. One superepoch is illustrated in Figure 5. We now describe a superepoch formally.

Start with two networks $\underline{\pi}$ and $\overline{\pi}$, both initialized to π_0: $\underline{\pi}_0 = \overline{\pi}_0 = \pi_0$. Roll out batch $\mathcal{D}_0$ using $\underline{\pi}_0$ and train both networks on it with the PPO objective for one epoch, so that they are now both identical to $\pi_1 = \underline{\pi}_1 = \overline{\pi}_1$. Next, roll out batch $\mathcal{D}_1$ with $\underline{\pi}_1$ for both networks. Train the first network on $\mathcal{D}_1$ using the PPO objective, obtaining $\underline{\pi}_2$. Train the second network on $\mathcal{D}_0 \cup \mathcal{D}_1$ using RAGGER, obtaining $\overline{\pi}_2$. This is the first epoch at which the two methods differ.

Then, for $i = 2, 3, 4$, roll out batch $\underline{\mathcal{D}}_i$ with $\underline{\pi}_i$ and batch $\overline{\mathcal{D}}_i$ with $\overline{\pi}_i$. As before, train the first network on $\underline{\mathcal{D}}_i$ using PPO, obtaining $\underline{\pi}_{i+1}$, and train the second network on $\overline{\mathcal{D}}_{i-1} \cup \overline{\mathcal{D}}_i$ using RAGGER (with $\overline{\mathcal{D}}_1 = \mathcal{D}_1$), obtaining $\overline{\pi}_{i+1}$. Having obtained $\underline{\pi}_5$ and $\overline{\pi}_5$, we either assign $\underline{\pi}_0, \overline{\pi}_0 \leftarrow \underline{\pi}_5$ or



(a) Local forgetting within a superepoch. One superepoch consists of 32 updates to the policy after π_1. The plot shows the forgetting metric for PPO and RAGGER over these 32 updates averaged across all superepochs.



(b) We plot the variation of local forgetting across training time. Each curve shows the average local forgetting metric at a certain number of epochs after π_1 for PPO and RAGGER over all superepochs through the training process.

Figure 5: Local forgetting in PPO and RAGGER. Negative values indicate forgetting. It is clear that RAGGER experiences significantly less local forgetting than PPO at each update and across the training process.

$\underline{\pi}_0, \overline{\pi}_0 \leftarrow \overline{\pi}_5$ depending on whether our experiment is to test the PPO or RAGGER regime. We then loop back to the second step to perform the next superepoch.

An algorithm for this process is outlined in Algorithm 2. In summary, the experiment allows PPO and RAGGER to repeatedly drift but brings them back together every five epochs.

Local forgetting metric. Denote by $J(\pi_{\text{old}}, \pi, \mathcal{D})$ the value of the clipped surrogate objective of policy π with respect to the old policy π_{old} on dataset $\mathcal{D}$ rolled out by π_{old}. We define the local forgetting metric $\mathcal{F}$ at a given superepoch with initialization π_0 and current policy π (either from PPO or RAGGER updates) as

$$\mathcal{F}(\pi) := J(\pi_0, \pi, \mathcal{D}_0) - J(\pi_0, \pi_1, \mathcal{D}_0). \tag{8}$$

$\mathcal{F}(\pi)$ measures how much the surrogate objective on the first rollout $\mathcal{D}_0$ has changed after PPO updates yield policy π, with (larger) negative values indicating (more) forgetting.

The plot of the forgetting metrics for the PPO and RAGGER networks per superepoch and across superepochs is shown in Figure 5. It is easy to see that RAGGER experiences significantly less local forgetting than PPO. In particular, Figure 5(a) shows that the rate of forgetting as a function of more updates is much lower for RAGGER than for PPO on average across training. Figure 5(b) shows that local forgetting is small and does not deteriorate over the course of training for RAGGER, while PPO's local forgetting is substantially higher and can also worsen as training progresses.

5 Related Work

Massively parallel on-policy RL. Most prior work attributes PPO's plateau in massively parallel training to insufficient data diversity. DexPBT (Petrenko et al., 2023) trains a population of PPO

agents with population-based training, and EPO (Wang et al., 2025) uses a genetic algorithm over a population of agents. SAPG (Singla et al., 2024) splits the environments among a leader and several follower policies and trains the leader on the followers' data as well as its own; CPO (Shitanda et al., 2026) argues that SAPG's followers become too diverse and constrains them to stay near the leader. Yin et al. (2026) instead ease exploration by resetting episodes to a diverse set of start states.

Two recent papers identify causes other than exploration: Beukman et al. (2026) argue that updates can be too large relative to noise in the sampled objective, and Bharthulwar et al. (2025) show that synchronized environment resets introduce nonstationarity that destabilizes training. Our findings are complementary; PPO often learns good behavior and then loses it, and a single policy that reuses its previous rollout outperforms "leader-follower" methods (Sec. 4.1). Since RAGGER changes only the PPO update, it can be combined with populations, leader-follower schemes, and reset strategies.

Data reuse in on-policy methods. Several methods use PPO-style updates on data from other policies, primarily to improve sample efficiency in small batch settings (Han & Sung, 2018; Queeney et al., 2021; Meng et al., 2023). Closest to ours, AMBER (Han & Sung, 2018) trains PPO on the current batch together with several past batches, computing each sample's ratio with respect to the policy that collected it. However, we study reusing the previous rollout in a regime where data is *abundant*, and find that its main effect is to prevent PPO's collapse. Reuse is not beneficial in itself: OpenAI et al. (2019) report that reusing stale data can slow learning, and in our ablations, extra passes over each sample does not reproduce RAGGER's gains (Sec. 4.3).

Instability and forgetting in policy optimization. Neural networks trained on a changing data distribution tend to forget what they learned earlier (McCloskey & Cohen, 1989; French, 1999). A common remedy is replay, which interleaves past examples with new ones (Chaudhry et al., 2019); in RL, replay is used against forgetting when an agent learns a sequence of tasks (Rolnick et al., 2019).

On-policy RL is non-stationary even within a single task, since the data distribution shifts as the policy changes. PPO's clipping neither strictly bounds the probability ratio nor enforces a well-defined trust region (Wang et al., 2020), and its behavior depends heavily on implementation details (Engstrom et al., 2020; Andrychowicz et al., 2020). Dohare et al. (2024) describe plasticity loss in PPO, with the performance on a locomotion task rising for 3 million steps and then collapsing; Moalla et al. (2024) connect this to a loss of feature rank that undermines the trust region. Their remedies modify the optimizer or add representation losses. RAGGER keeps PPO's network, optimizer, and loss, and changes only which data each update sees; within a single task, this suffices to prevent PPO's collapse.

6 Conclusion and Limitations

We introduced RAGGER, a minimal modification to PPO that trains each update on the previous rollout as well as the new one. On long-horizon dexterous manipulation, this single change prevents the collapse of standard PPO and outperforms SAPG and CPO on all four AllegroKuka tasks; it also improves PPO on hand-object tracking. Our ablations indicate that the gain comes from continuing to optimize the previous rollout's surrogate objective, and our local-forgetting experiments further suggest that the previous rollout acts as a regularizer, limiting how much each update degrades the objective on recent rollouts. More broadly, our results suggest that on-policy RL at scale can be limited as much by stability as by exploration.

Limitations. First, we lack a theoretical explanation for why optimizing the previous surrogate prevents collapse. Second, carrying exactly one previous rollout is a heuristic; we have not studied how many rollouts to carry. On harder tasks, one rollout may not be enough to prevent collapse, and it remains open whether carrying more would help. Third, we tuned RAGGER's hyperparameters very little: it uses PPO's settings, except that it doubles the minibatch size to cover the aggregated batch (App. B.2). Finally, since RAGGER changes only PPO's update, it can be combined with population-based methods such as DexPBT (Petrenko et al., 2023) and EPO (Wang et al., 2025), or with leader-follower methods; we have not yet tested these combinations.

AI USE STATEMENT

In this work, we used generative AI tools for feedback on our experimental design and to implement our code (Claude Code). We have not used generative AI tools to develop theoretical models or conceptual frameworks, propose or refine hypotheses, or interpret results, and generating synthetic data, formulating mathematical claims or proofs, translation, cleaning or reformatting datasets, and qualitative or thematic data analysis are not applicable to this work. Additionally, we used generative AI tools (Claude, ChatGPT) to polish the text, check novelty against prior work, search for information, and create or modify figures. We validated the AI-written code and drew all conclusions from the results ourselves. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

RAGGER is specified by its objective (Sec. 3) and Algorithm 1, and App. B describes the remaining implementation details, including the learning-rate controller. We implement it on top of the public PPO implementation in `rl_games` (Makoviichuk & Makoviychuk, 2021). All tasks are publicly available: the Isaac Gym and AllegroKuka tasks (Makoviychuk et al., 2021; Petrenko et al., 2023), and the Dexmachina tracking tasks, which we run with their original curriculum and hyperparameters (Mandi et al., 2025). For SAPG and CPO, we run the official implementations with their published settings. App. B.2 lists the hyperparameters of every method on every task. Our code is included in the supplementary material and will be made public upon publication.

REFERENCES

Marcin Andrychowicz, Anton Raichuk, Piotr Stańczyk, Manu Orsini, Sertan Girgin, et al. What matters in on-policy reinforcement learning? a large-scale empirical study, 2020. arXiv:2006.05990.

Michael Beukman, Khimya Khetarpal, Zeyu Zheng, Will Dabney, Jakob Foerster, et al. Preventing learning stagnation in ppo by scaling to 1 million parallel environments, 2026. arXiv:2603.06009.

Sid Bharthulwar, Stone Tao, and Hao Su. Staggered environment resets improve massively parallel on-policy reinforcement learning, 2025. arXiv:2511.21011.

Arslan Chaudhry, Marcus Rohrbach, Mohamed Elhoseiny, Thalaiyasingam Ajanthan, Puneet K. Dokania, et al. On tiny episodic memories in continual learning, 2019. arXiv:1902.10486.

Shibhansh Dohare, J. Fernando Hernandez-Garcia, Qingfeng Lan, Parash Rahman, Ashique Rupam Mahmood, et al. Loss of plasticity in deep continual learning. *Nature*, 632:768 – 774, 2024. URL https://api.semanticscholar.org/CorpusID:259251905.

Logan Engstrom, Andrew Ilyas, Shibani Santurkar, Dimitris Tsipras, Firdaus Janoos, et al. Implementation matters in deep policy gradients: A case study on ppo and trpo, 2020. arXiv:2005.12729.

Zicong Fan, Omid Taheri, Dimitrios Tzionas, Muhammed Kocabas, Manuel Kaufmann, et al. Arctic: A dataset for dexterous bimanual hand-object manipulation, 2023. arXiv:2204.13662.

Robert M. French. Catastrophic forgetting in connectionist networks. *Trends in Cognitive Sciences*, 3(4): 128–135, 1999. ISSN 1364-6613. doi: https://doi.org/10.1016/S1364-6613(99)01294-2. URL https://www.sciencedirect.com/science/article/pii/S1364661399012942.

Genesis. The role of simulation in scalable robotics, genesis world 1.0, and the path forward. *Genesis AI Blog*, May 2026. URL https://www.genesis.ai/blog/the-role-of-simulation-in-scalable-robotics-genesis-world-10-and-the-path-forward.

Seungyul Han and Youngchul Sung. Amber: Adaptive multi-batch experience replay for continuous action control, 2018. arXiv:1710.04423.

Jacky Liang, Viktor Makoviychuk, Ankur Handa, Nuttapong Chentanez, Miles Macklin, et al. Gpu-accelerated robotic simulation for distributed reinforcement learning. In Aude Billard, Anca Dragan, Jan Peters, and Jun Morimoto (eds.), *Proceedings of The 2nd Conference on Robot Learning*, volume 87 of *Proceedings of Machine Learning Research*, pp. 270–282. PMLR, 29–31 Oct 2018. URL https://proceedings.mlr.press/v87/liang18a.html.

Denys Makoviichuk and Viktor Makoviychuk. rl-games: A high-performance framework for reinforcement learning. https://github.com/Denys88/rl_games, May 2021.

Viktor Makoviychuk, Lukasz Wawrzyniak, Yunrong Guo, Michelle Lu, Kier Storey, et al. Isaac gym: High performance gpu-based physics simulation for robot learning, 2021. arXiv:2108.10470.

Zhao Mandi, Yifan Hou, Dieter Fox, Yashraj Narang, Ajay Mandlekar, et al. Dexmachina: Functional retargeting for bimanual dexterous manipulation, 2025. arXiv:2505.24853.

Walter Mayor, Johan Obando-Ceron, Aaron Courville, and Pablo Samuel Castro. The impact of on-policy parallelized data collection on deep reinforcement learning networks, 2025. arXiv:2506.03404.

Michael McCloskey and Neal J. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. volume 24 of *Psychology of Learning and Motivation*, pp. 109–165. Academic Press, 1989. doi: https://doi.org/10.1016/S0079-7421(08)60536-8. URL https://www.sciencedirect.com/science/article/pii/S0079742108605368.

Wenjia Meng, Qian Zheng, Gang Pan, and Yilong Yin. Off-policy proximal policy optimization. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. ISBN 978-1-57735-880-0. doi: 10.1609/aaai.v37i8.26099. URL https://doi.org/10.1609/aaai.v37i8.26099.

Skander Moalla, Andrea Miele, Daniil Pyatko, Razvan Pascanu, and Caglar Gulcehre. No representation, no trust: Connecting representation, collapse, and trust issues in ppo, 2024. arXiv:2405.00662.

NVIDIA, Mayank Mittal, Pascal Roth, James Tigue, Antoine Richard, et al. Isaac lab: A gpu-accelerated simulation framework for multi-modal robot learning, 2025. arXiv:2511.04831.

OpenAI, Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, et al. Dota 2 with large scale deep reinforcement learning, 2019. arXiv:1912.06680.

Fabio Pardo, Arash Tavakoli, Vitaly Levdik, and Petar Kormushev. Time limits in reinforcement learning. In Jennifer Dy and Andreas Krause (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4045–4054. PMLR, 10–15 Jul 2018. URL https://proceedings.mlr.press/v80/pardo18a.html.

Aleksei Petrenko, Arthur Allshire, Gavriel State, Ankur Handa, and Viktor Makoviychuk. Dexpbt: Scaling up dexterous manipulation for hand-arm systems with population based training, 2023. arXiv:2305.12127.

James Queeney, Ioannis Ch. Paschalidis, and Christos G. Cassandras. Generalized proximal policy optimization with sample reuse, 2021. arXiv:2111.00072.

David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Greg Wayne. Experience replay for continual learning, 2019. arXiv:1811.11682.

John Schulman, Sergey Levine, Philipp Moritz, Michael I. Jordan, and Pieter Abbeel. Trust region policy optimization, 2017a. arXiv:1502.05477.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017b. arXiv:1707.06347.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation, 2018. arXiv:1506.02438.

Naoki Shitanda, Motoki Omura, Tatsuya Harada, and Takayuki Osa. Rethinking policy diversity in ensemble policy gradient in large-scale reinforcement learning, 2026. arXiv:2603.01741.

Jayesh Singla, Ananye Agarwal, and Deepak Pathak. Sapg: Split and aggregate policy gradients, 2024. arXiv:2407.20230.

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 (eds.), *Advances in Neural Information Processing Systems*, volume 12. MIT Press, 1999. URL https://proceedings.neurips.cc/paper_files/paper/1999/file/464d828b85b0bed98e80ade0a5c43b0f-Paper.pdf.

Jianren Wang, Yifan Su, Abhinav Gupta, and Deepak Pathak. Evolutionary policy optimization, 2025. arXiv:2503.19037.

Yuhui Wang, Hao He, Chao Wen, and Xiaoyang Tan. Truly proximal policy optimization, 2020. arXiv:1903.07940.

Patrick Yin, Tyler Westenbroek, Zhengyu Zhang, Joshua Tran, Ignacio Dagnino, et al. Emergent dexterity via diverse resets and large-scale reinforcement learning, 2026. arXiv:2603.15789.

Kevin Zakka, Qiayuan Liao, Brent Yi, Louis Le Lay, Koushil Sreenath, et al. mjlab: A lightweight framework for gpu-accelerated robot learning, 2026. arXiv:2601.22074.

Xinghao Zhu, Zixi Liu, Shalin Jain, Chenran Li, Milad Noori, et al. Learning dexterous manipulation using contact wrench guidance from human demonstration, 2026. arXiv:2607.00033.

APPENDIX

TABLE OF CONTENTS

A Additional Background **13**

A.1 Policy Gradient Methods 13

B Additional Details **14**

B.1 Algorithm 14

B.2 Hyperparameters 14

B.3 Environments 17

C Additional Local Forgetting Results **18**

A ADDITIONAL BACKGROUND

A.1 POLICY GRADIENT METHODS

We consider the usual reinforcement learning setting where each task is modeled as an infinite-horizon discounted MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \rho, \gamma)$, with state space $\mathcal{S}$, action space $\mathcal{A}$, transition function $\mathcal{P} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$[2], reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, initial-state distribution ρ, and discount factor $\gamma \in [0, 1)$. The objective is to find a policy $\pi : \mathcal{S} \to \Delta(\mathcal{A})$ that maximizes the expected cumulative discounted return from $t = 1$ to ∞:

$$\pi^* = \arg\max_\theta J(\pi_\theta), \tag{9}$$

where for policy π, we define

$$J(\pi) = \mathbb{E}_{s_1 \sim \rho, a_t \sim \pi(s_t), s_{t+1} \sim \mathcal{P}(s_t, a_t)} \left[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) \right]. \tag{10}$$

Given a policy π, we define the state-value function $V^\pi(s) := \mathbb{E}_\pi[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) | s_1 = s]$ and action-value function $Q^\pi(s, a) := \mathbb{E}_\pi[\sum_{t=1}^{\infty} \gamma^{t-1} r(s_t, a_t) | s_1 = s, a_1 = a]$. The *advantage* $A^\pi(s, a) := Q^\pi(s, a) - V^\pi(s)$ measures how much better the action a is than the policy's average action at that state s. Policy gradient methods optimize $J(\pi)$ by iteratively estimating the gradient of the objective with respect to the policy parameters θ and performing gradient ascent. The policy gradient theorem (Sutton et al., 1999) provides a way to compute this gradient:

$$\nabla_\theta J(\pi_\theta) = \frac{1}{1-\gamma} \mathbb{E}_{s \sim d^{\pi_\theta}, a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(a|s) A^{\pi_\theta}(s, a) \right], \tag{11}$$

where d^π is the discounted state visitation distribution under policy π. In practice, following TRPO (Schulman et al., 2017a), one usually optimizes the surrogate objective

$$J^{\text{sur}}(\pi_{\theta_{\text{old}}}, \pi_\theta) = \mathbb{E}_{s \sim d^{\pi_{\theta_{\text{old}}}}, a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} A^{\pi_{\text{old}}}(s, a) \right] \tag{12}$$

over θ with $\pi_{\theta_{\text{old}}}$ collecting data. This objective is consistent with the policy gradient theorem: one can verify that $\nabla_\theta J^{\text{sur}}(\pi_{\theta_{\text{old}}}, \pi_\theta) = (1-\gamma) \nabla_\theta J(\pi_\theta)$ at $\theta = \theta_{\text{old}}$ and thus may use the surrogate objective to compute the policy gradient. In practice, one uses a finite batch of trajectories $\mathcal{D}$ to estimate the expectation and gradient:

$$\hat{J}(\pi_{\text{old}}, \pi_\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,a) \in \mathcal{D}} \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)} \hat{A}^{\pi_{\theta_{\text{old}}}}(s, a), \tag{13}$$

$$\theta \leftarrow \theta_{\text{old}} + \eta \nabla_\theta \hat{J}(\pi_{\text{old}}, \pi_\theta). \tag{14}$$

[2]Here, $\Delta(X)$ denotes the set of all probability distributions over set X.

Here the empirical advantage $\hat{A}^{\pi_{\theta_{\text{old}}}}(s,a)$ is typically computed using generalized advantage estimation (GAE) (Schulman et al., 2018) (so we also estimate a value function). TRPO finds that optimizing the surrogate $\hat{J}$ requires some care, since improvement is only guaranteed within a *trust region* around the old policy $\pi_{\theta_{\text{old}}}$ enforced by a total-variation/KL-divergence constraint, e.g. $\max_s D_{\text{KL}}[\pi_{\text{old}}(\cdot|s)||\pi(\cdot|s)] \leq \delta$ (Schulman et al., 2017a). Instead of optimizing the objective constrained to be within the trust region, Proximal Policy Optimization (Schulman et al., 2017b) proposes a clipped surrogate objective that forces small policy updates when the probability ratio $\rho_\theta(a|s) := \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ is prone to go outside the trust region:

$$\hat{J}^{\text{PPO}}(\pi_{\text{old}}, \pi_\theta) = \frac{1}{|\mathcal{D}|} \sum_{(s,a)\in\mathcal{D}} \min\left(\rho_\theta(a|s)\hat{A}^{\pi_{\theta_{\text{old}}}}(s,a), \text{clip}\left(\rho_\theta(a|s), 1-\epsilon, 1+\epsilon\right)\hat{A}^{\pi_{\theta_{\text{old}}}}(s,a)\right). \tag{15}$$

Here $\epsilon \sim 0.2$ is a clipping hyperparameter. The KL constraint continues to be enforced in soft fashion, with a learning rate controller that reduces (increases) the learning rate when the (mean) KL divergence across the batch of trajectories exceeds (falls below) a target value.

PPO has been shown to be a simple and effective policy gradient method that is widely used in practice, especially in large-scale RL training (Makoviychuk et al., 2021).

B Additional Details

B.1 Algorithm

The learning rate controller (Makoviichuk & Makoviychuk, 2021) is modified to compute the mean KL divergence across the aggregated dataset $\mathcal{D}_{k-1} \cup \mathcal{D}_k$; in particular, we compute

$$d_{\text{KL}}^{\pi_{k-1}\|\pi_k}(\theta) = \frac{1}{|\mathcal{D}_{k-1}| + |\mathcal{D}_k|} \left(\sum_{(s,a)\in\mathcal{D}_{k-1}} D_{\text{KL}}\left[\pi_\theta(\cdot|s) \parallel \pi_{k-1}(\cdot|s)\right] + \sum_{(s,a)\in\mathcal{D}_k} D_{\text{KL}}\left[\pi_\theta(\cdot|s) \parallel \pi_k(\cdot|s)\right] \right) \tag{16}$$

and set LR $\eta \leftarrow \eta/1.5$ if $d_{\text{KL}}^{\pi_{k-1}\|\pi_k}(\theta) > 2d_{\text{target}}$ and $\eta \leftarrow 1.5\eta$ if $d_{\text{KL}}^{\pi_{k-1}\|\pi_k}(\theta) < d_{\text{target}}/2$.

Measuring the KL against the collecting policy matches the baselines: in SAPG, for instance, the leader's KL on a follower's data is measured against that follower's policy. In preliminary experiments, the adaptive controller improved performance over a fixed learning rate for every method, so we use it for all methods.

B.2 Hyperparameters

B.2.1 AllegroKuka Tasks

Table 2 lists the training hyperparameters of PPO and PPO+RAGGER on the four AllegroKuka tasks (`Regrasping`, `Reorientation`, `Throw` and `Two-Arms Reorientation`). The same values are used on all four tasks and for all seeds. PPO+RAGGER trains every epoch on the $H_{\text{buf}} = 16$ carried steps of the previous rollout together with the 16 fresh steps, so the mini-batch size are twice PPO's while the number of gradient updates per epoch is the same.

Table 3 lists the corresponding hyperparameters for the two in-hand reorientation tasks (Shadow Hand and Allegro Hand), which use MLP policies. PPO+RAGGER again trains on the $H_{\text{buf}} = 8$ carried steps of the previous rollout together with the 8 fresh steps.

Tables 4 and 5 list the hyperparameters of the SAPG and CPO baselines on the AllegroKuka and in-hand reorientation tasks, using the fields of Shitanda et al. (2026). We run the official implementations with the settings reported there. One of our five CPO seeds failed to learn on Shadow Hand, which accounts for CPO's large standard deviation on that task in Table 1. On Two-Arms Reorientation, our SAPG score (3.98) is far below the 28.58 reported by Singla et al. (2024), but consistent with the reproduction by Shitanda et al. (2026) (5.11). This gap is likely due to the original run lasting 5×10^{10} frames rather than 2×10^{10}. Even so, RAGGER's score after 2×10^{10} frames (44.13) exceeds the originally reported 28.58.

Table 2: Training hyperparameters for the AllegroKuka tasks (Regrasping, Reorientation, Throw and Two-Arms Reorientation), identical across tasks and seeds.

Hyperparameter	PPO	PPO+RAGGER
Number of environments, N	24,576	24,576
Rollout length, H_{buf}	16	16
Discount factor, γ	0.99	0.99
GAE smoothing factor, λ	0.95	0.95
LSTM hidden units	768	768
MLP hidden layers	[768, 512, 256]	[768, 512, 256]
Activation	ELU	ELU
Learning rate (initial)	1×10^{-4}	1×10^{-4}
KL target for the learning-rate controller, d_{target}	0.016	0.016
Gradient-norm clipping	1.0	1.0
Clipping factor, ϵ	0.1	0.1
Entropy coefficient	0	0
Critic coefficient, c_V	4.0	4.0
Bounds loss coefficient	1×10^{-4}	1×10^{-4}
Reward scale	0.01	0.01
Observation, value and advantage normalization	yes	yes
Mini-epochs, E	2	2
Gradient updates per epoch	8	8
Carried steps per epoch (previous rollout)	–	16
Training window / LSTM sequence length	16	32
Mini-batch size, B	98,304 ($4N$)	196,608 ($8N$)
Value-function clipping	yes	no

Table 3: Training hyperparameters for the in-hand reorientation tasks (Shadow Hand and Allegro Hand), identical across seeds. The two tasks differ only in the MLP size.

Hyperparameter	PPO	PPO+RAGGER
Number of environments, N	24,576	24,576
Rollout length, H_{buf}	8	8
Discount factor, γ	0.99	0.99
GAE smoothing factor, λ	0.95	0.95
MLP hidden layers (Shadow Hand)	[512, 512, 256, 128]	[512, 512, 256, 128]
MLP hidden layers (Allegro Hand)	[512, 256, 128]	[512, 256, 128]
Activation	ELU	ELU
Learning rate (initial)	5×10^{-4}	5×10^{-4}
KL target for the learning-rate controller, d_{target}	0.016	0.016
Gradient-norm clipping	1.0	1.0
Clipping factor, ϵ	0.2	0.2
Entropy coefficient	0	0
Critic coefficient, c_V	4.0	4.0
Bounds loss coefficient	1×10^{-4}	1×10^{-4}
Reward scale	0.01	0.01
Observation, value and advantage normalization	yes	yes
Mini-epochs, E	5	5
Gradient updates per epoch	10	10
Carried steps per epoch (previous rollout)	–	8
Training window	8	16
Mini-batch size, B	98,304 ($4N$)	196,608 ($8N$)
Value-function clipping	yes	no

Table 4: Training hyperparameters of SAPG and CPO on the AllegroKuka tasks, with the fields of Shitanda et al. (2026). The upper section lists hyperparameters shared by SAPG and CPO; the lower section lists those specific to CPO. Reorientation and Two-Arms Reorientation use the same values. Both methods use $N = 24{,}576$ environments, split into six blocks of 4,096 (one leader and five followers).

Hyperparameter	Regrasping	Reorientation / Two-Arms Reorientation	Throw
Common hyperparameters (SAPG and CPO)			
Discount factor, γ	0.99	0.99	0.99
GAE smoothing factor, λ	0.95	0.95	0.95
LSTM hidden units	768	768	768
MLP hidden layers	[768, 512, 256]	[768, 512, 256]	[768, 512, 256]
Learning rate	1×10^{-4}	1×10^{-4}	1×10^{-4}
KL threshold for LR update	0.016	0.016	0.016
Grad norm	1.0	1.0	1.0
Entropy coefficient	0	0.005	0
Clipping factor, ϵ	0.1	0.1	0.1
Mini-batch size	$4N$	$4N$	$4N$
Critic coefficient	4.0	4.0	4.0
Horizon length	16	16	16
LSTM sequence length	16	16	16
Bounds loss coefficient	1×10^{-4}	1×10^{-4}	1×10^{-4}
Mini epochs	2	2	2
CPO-specific hyperparameters			
β (Eq. 11 in Shitanda et al., 2026)	0.001	0.001	0.001
Forward KL constraint temperature, λ_f	0.2	0.2	0.1
Reverse KL constraint temperature, λ_r	0	0	0
Adversarial reward scaling factor, λ_{adv}	0	0	0

Table 5: Training hyperparameters of SAPG and CPO on the in-hand reorientation tasks, with the fields of Shitanda et al. (2026). The upper section lists hyperparameters shared by SAPG and CPO; the lower section lists those specific to CPO. Both methods use $N = 24{,}576$ environments, split into six blocks of 4,096 (one leader and five followers).

Hyperparameter	Shadow Hand	Allegro Hand
Common hyperparameters (SAPG and CPO)		
Discount factor, γ	0.99	0.99
GAE smoothing factor, λ	0.95	0.95
MLP hidden layers	[512, 512, 256, 128]	[512, 256, 128]
Learning rate	5×10^{-4}	5×10^{-4}
KL threshold for LR update	0.016	0.016
Grad norm	1.0	1.0
Entropy coefficient	0.005	0
Clipping factor, ϵ	0.2	0.2
Mini-batch size	$4N$	$4N$
Critic coefficient	4.0	4.0
Horizon length	8	8
Bounds loss coefficient	1×10^{-4}	1×10^{-4}
Mini epochs	5	5
CPO-specific hyperparameters		
β (Eq. 11 in Shitanda et al., 2026)	0.001	0.0005
Forward KL constraint temperature, λ_f	0.2	0.1
Reverse KL constraint temperature, λ_r	0	0
Adversarial reward scaling factor, λ_{adv}	0.01	0.001

B.2.2 HAND-OBJECT TRACKING TASKS

Table 6 lists the training hyperparameters of PPO and PPO+RAGGER on the four DexMachina hand-object tracking tasks (`Ketchup`, `Waffle Iron`, `Notebook` and `Mixer`, all with the Inspire hands), identical across tasks and seeds. Both methods keep the DexMachina training defaults. PPO+RAGGER trains every epoch on the $H_{\text{buf}} = 32$ carried steps of the previous rollout together with the 32 fresh steps.

Table 6: Training hyperparameters for the hand-object tracking tasks (Ketchup, Waffle Iron, Notebook and Mixer), identical across tasks and seeds.

Hyperparameter	PPO	PPO+RAGGER
Number of environments, N	12,000	12,000
Rollout length, H_{buf}	32	32
Discount factor, γ	0.99	0.99
GAE smoothing factor, λ	0.95	0.95
MLP hidden layers	[512, 512, 256, 128]	[512, 512, 256, 128]
Activation	ELU	ELU
Learning rate (initial)	3×10^{-4}	3×10^{-4}
KL target for the learning-rate controller, d_{target}	0.008	0.008
Gradient-norm clipping	1.0	1.0
Clipping factor, ϵ	0.2	0.2
Entropy coefficient	0	0
Critic coefficient, c_V	4.0	4.0
Bounds loss coefficient	1×10^{-4}	1×10^{-4}
Reward scale	0.1	0.1
Observation normalization	no	no
Value and advantage normalization	yes	yes
Mini-epochs, E	14	14
Gradient updates per epoch	56	56
Carried steps per epoch (previous rollout)	–	32
Training window	32	64
Mini-batch size, B	96,000 ($8N$)	192,000 ($16N$)
Value-function clipping	yes	yes

B.3 ENVIRONMENTS

Figure 6 shows the six Isaac Gym tasks of Section 4.1, and Figure 7 the four DexMachina hand-object tracking tasks of Section 4.2.

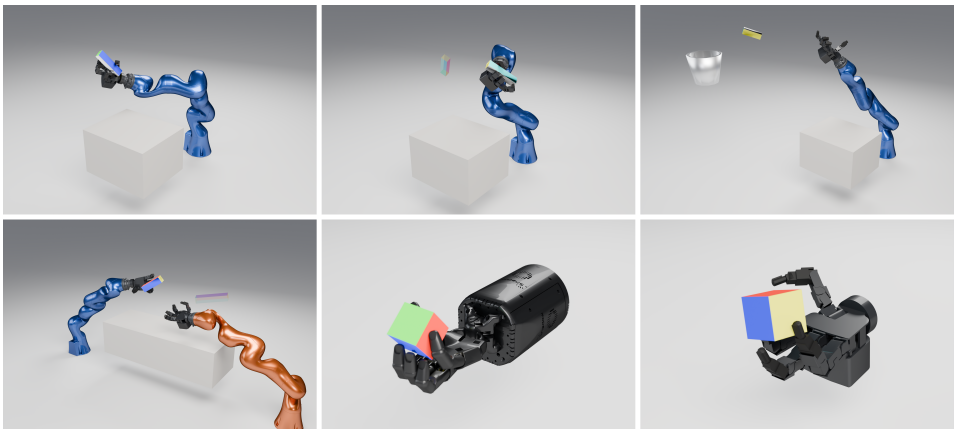


Figure 6: The six evaluation tasks. Top row and bottom left: the four AllegroKuka tasks (Regrasping, Reorientation, Throw, Two-Arms Reorientation). Bottom middle and right: Shadow Hand and Allegro Hand.

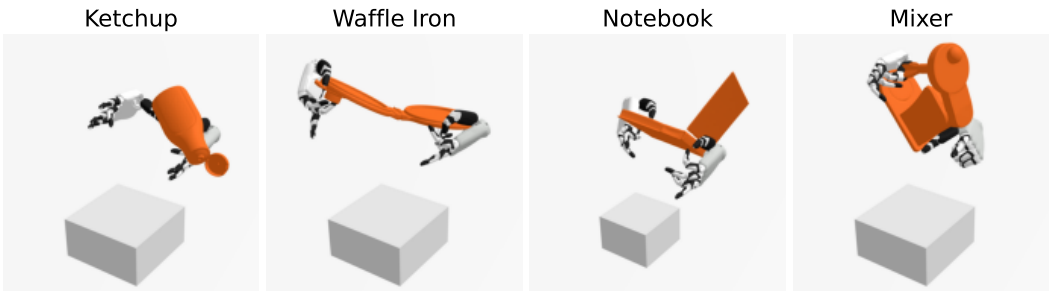


Figure 7: The four hand-object tracking tasks from DexMachina (Mandi et al., 2025): Ketchup, Waffle Iron, Notebook, and Mixer. In each, two dexterous hands track a human demonstration of bimanual manipulation of an articulated object.

Algorithm 2 One superepoch of the local forgetting experiment. Blue objects represent working with the PPO network and Violet objects for the RAGGER network likewise. (in particular $\pi_i \neq \pi_i$).

Input: Initial network π_0

$\mathcal{D}_0 \leftarrow \text{roll}(\pi_0)$; $\pi_1 \leftarrow \text{PPO}(\pi_0, \mathcal{D}_0)$; $\mathcal{D}_1 \leftarrow \text{roll}(\pi_1)$ — Both networks equal $= \pi_1$

$\pi_2 \leftarrow \text{PPO}(\pi_1, \mathcal{D}_1)$; $\pi_2 \leftarrow \text{RAgger}(\pi_1, \mathcal{D}_0 \cup \mathcal{D}_1)$ — Networks diverge!

for $i = 2, 3, 4$ **do**

  $\mathcal{D}_i \leftarrow \text{roll}(\pi_i)$; $\pi_{i+1} \leftarrow \text{PPO}(\pi_i, \mathcal{D}_i)$

  $\mathcal{D}_i \leftarrow \text{roll}(\pi_i)$; $\pi_{i+1} \leftarrow \text{RAgger}(\pi_i, \mathcal{D}_{i-1} \cup \mathcal{D}_i)$

for $i = 1, \ldots, 4$ **do** — Compute local forgetting metric:

  $\mathcal{F}_i \leftarrow J(\pi_0, \pi_{i+1}, \mathcal{D}_0) - J(\pi_0, \pi_1, \mathcal{D}_0)$ — for PPO

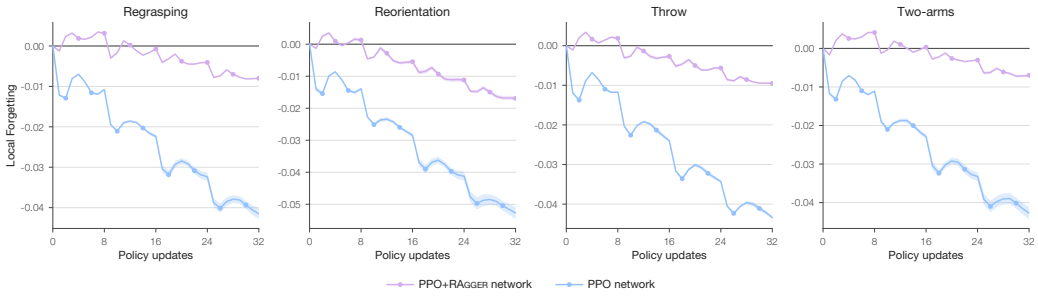
  $\mathcal{F}_i \leftarrow J(\pi_0, \pi_{i+1}, \mathcal{D}_0) - J(\pi_0, \pi_1, \mathcal{D}_0)$ — for RAGGER

$\pi_0 \leftarrow \pi_5$ — Reset superepoch start state

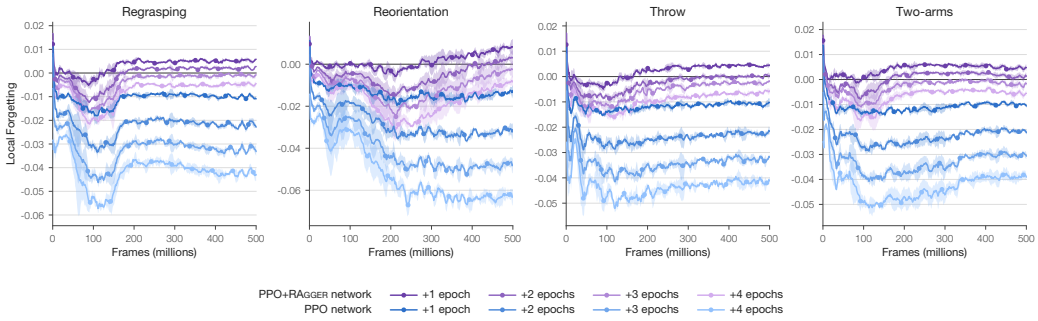
C ADDITIONAL LOCAL FORGETTING RESULTS

We continue the analysis of local forgetting in PPO and RAGGER on the AllegroKuka tasks begun in Section 4.4. Algorithm 2 describes a superepoch.

It can be asked whether the results in Figure 5 are due to the PPO policy being used as the common branch through training; to test this, we ran the same experiment but merged the policies to the RAGGER policy at the end of each superepoch instead of the PPO policy. Figure 8 shows this counterpart to Figure 5 when superepochs assign the RAGGER policy to both PPO and RAGGER at the end of each superepoch instead of the PPO policy. The results are almost identical to Figure 5, showing that the local forgetting effect is not due to the PPO policy being used as the starting point for both methods but is instead a property of the methods themselves.



(a) Identical to Figure 5(a) but for policies merged to the RAGGER policy after each superepoch.



(b) Identical to Figure 5(b) but for policies merged to the RAGGER policy after each superepoch.

Figure 8: The counterpart to Figure 5 when superepochs assign the RAGGER policy to both PPO and RAGGER at the end of each superepoch instead of the PPO policy. The results are almost identical to Figure 5, showing that the local forgetting effect is a property of the methods themselves.