On-Policy Policy Gradient Reinforcement Learning Without On-Policy Sampling

Nicholas E. Corrado[1], **Josiah P. Hanna**[1]
ncorrado@wisc.edu, jphanna@cs.wisc.edu

[1]**Department of Computing Sciences, University of Wisconsin–Madison**

Abstract

On-policy reinforcement learning (RL) algorithms are typically characterized as algorithms that perform policy updates using i.i.d. trajectories collected by the agent's current policy. However, after observing only a finite number of trajectories, on-policy sampling may produce data that fails to match the expected on-policy data distribution. This *sampling error* leads to high-variance gradient estimates and data inefficient on-policy learning. Recent work in policy evaluation has shown that non-i.i.d., off-policy sampling can produce data with lower sampling error w.r.t. the expected on-policy distribution than on-policy sampling can produce (Zhong et al., 2022). Motivated by this observation, we introduce an adaptive, off-policy sampling method to reduce sampling error during on-policy policy gradient RL training. Our method, **P**roximal **R**obust **O**n-**P**olicy **S**ampling (PROPS), reduces sampling error by collecting data with a *behavior policy* that increases the probability of sampling actions that are under-sampled w.r.t. the current policy. We empirically evaluate PROPS on MuJoCo benchmark tasks and demonstrate that (1) PROPS decreases sampling error throughout training and (2) increases the data efficiency of on-policy policy gradient algorithms.

1 Introduction

One of the most widely used classes of reinforcement learning (RL) algorithms is the class of on-policy policy gradient algorithms. These algorithms optimize a parameterized policy via gradient ascent to increase the probability of observed actions with high expected returns under the current policy. In practice, the policy gradient is typically estimated using the Monte Carlo estimator, an average over i.i.d. trajectories sampled from the current policy. This estimator is unbiased and consistent: as the number of collected samples increases, the empirical distribution of data converges to the expected on-policy distribution, and thus the estimated gradient converges to the true gradient. However, with finite data, the empirical distribution of data often differs from the desired on-policy data distribution—a mismatch we call *sampling error*. Sampling error leads to inaccurate gradient estimates, high-variance updates, and potentially convergence to suboptimal policies.

With i.i.d. on-policy sampling, the only way to reduce sampling error is to collect more data. Alternatively, we can reduce sampling error more efficiently using adaptive, *off-policy* sampling. To illustrate, consider an MDP with two discrete actions A and B, and suppose the current policy π places equal probability on both actions in some state s. After 10 visits to s under π, we will observe both actions 5 times in expectation. Now suppose that after the first 9 visits to s, we actually observe A 4 times and B 5 times. If we sample an action from π upon our next visit to s, we may sample B, and our data will not match the expected on-policy distribution. Alternatively, if we select the under-sampled action A with probability 1, we will observe each action 5 times, making the aggregate data match the on-policy distribution even though this final action was sampled off-policy.

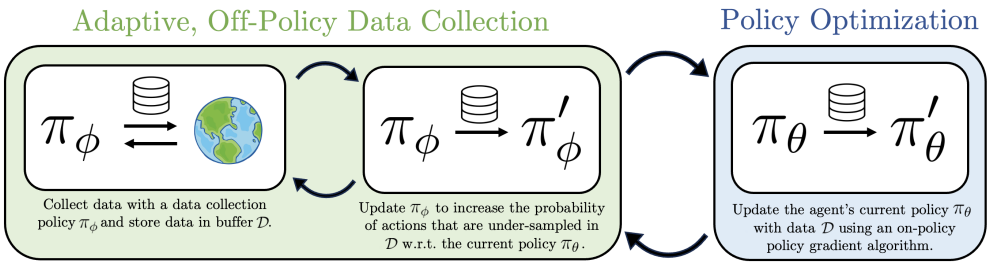


Figure 1: An overview of PROPS. We collect data with a separate data collection policy π_ϕ that we continually adapt to reduce sampling error in $\mathcal{D}$ with respect to the agent's current policy.

Recently, Zhong et al. (2022) introduced an adaptive, off-policy sampling method (ROS) that can produce data that more closely matches the on-policy distribution than data acquired through i.i.d. on-policy sampling. However, this work was limited to low-dimensional policy evaluation tasks. Moreover, ROS required large batches of data to reduce sampling error – approximately 5000 samples on tasks like CartPole-v1 (Brockman et al., 2016) – and struggled to reduce sampling error on tasks with continuous actions. To make adaptive sampling practical for data-efficient RL, we need methods that can reduce sampling error in high-dimensional continuous-action tasks while using the smaller batch sizes typically used in RL. These observations raise the following question: can on-policy policy gradient algorithms learn more efficiently *without* on-policy sampling?

In this work, we address these challenges and show for the first time that on-policy policy gradient algorithms are more data-efficient learners when they use on-policy data acquired with adaptive, off-policy sampling. Our method, **P**roximal **R**obust **O**n-**P**olicy **S**ampling (PROPS), adaptively corrects sampling error in previously collected data by increasing the probability of sampling actions that are under-sampled with respect to the current policy (Fig. 1). We empirically evaluate PROPS on continuous-action MuJoCo benchmark tasks and show that (1) PROPS reduces sampling error throughout training and (2) increases the data efficiency of on-policy policy gradient algorithms. In summary, our contributions are

1. We introduce PROPS, a scalable adaptive sampling algorithm for on-policy policy gradient learning that reduces sampling error w.r.t. the agent's current policy.
2. We demonstrate empirically that PROPS reduces sampling error more efficiently than on-policy sampling and ROS.
3. We show empirically that PROPS increases data efficiency in on-policy policy gradient RL.

2 Related Work

Prior works have used adaptive off-policy sampling to reduce sampling error in the policy evaluation subfield of RL. Zhong et al. (2022) first proposed that adaptive off-policy sampling could produce data that more closely matches the on-policy distribution than on-policy sampling could produce. Mukherjee et al. (2022) use a deterministic sampling rule to take actions in a particular proportion. Other works in the bandit setting use a non-adaptive exploration policy to collect additional data conditioned on previously collected data (Tucker & Joachims, 2022; Wan et al., 2022; Konyushova et al., 2021). Since these works only focus on policy evaluation, they do not have to contend with a changing on-policy distribution as our work does for the control setting.

Several prior works propose importance sampling methods (Precup, 2000) to reduce sampling error without collecting new data. In the RL setting, Hanna et al. (2021) showed that reweighting off-policy data according to an estimated behavior policy can correct sampling error and improve policy evaluation. Similar methods exist for temporal difference learning (Pavse et al., 2020) and policy evaluation in the bandit setting (Li et al., 2015; Narita et al., 2019). While these works reduce sampling error by reweighting existing data, our work reduces sampling error *during data collection*.

As we will discuss in Section 5, the method we introduce permits data collected in one iteration of policy optimization to be re-used in future iterations rather than discarded as typically done by on-policy algorithms. Prior work has attempted to avoid discarding data by combining off-policy and on-policy updates with separate loss functions or by using alternative gradient estimates (Wang et al., 2016; Gu et al., 2016; 2017; Fakoor et al., 2020; O'Donoghue et al., 2016; Queeney et al., 2021). In contrast, our method modifies the sampling distribution at each iteration so that the entire data set of past and newly collected data matches the expected distribution under the current policy.

3 Preliminaries

We formalize the RL environment as a finite-horizon Markov decision process (Puterman, 2014) with state space $\mathcal{S}$, action space $\mathcal{A}$, transition dynamics $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$, reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, initial state distribution d_0, and discount factor $\gamma \in [0, 1)$. We consider stochastic policies $\pi_{\boldsymbol{\theta}} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ parameterized by $\boldsymbol{\theta}$ and let $d_{\pi_{\boldsymbol{\theta}}} : \mathcal{S} \times \mathcal{A} \to [0, 1]$ denote the state-action visitation distribution of $\pi_{\boldsymbol{\theta}}$. The RL objective is to find a policy maximizing expected discounted return, $J(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{H} \gamma^t r(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$, where H is the horizon. We refer to the policy used for data collection as the *behavior policy* and the policy that optimizes the return as the *target policy*.

Policy gradient algorithms perform gradient ascent over policy parameters to maximize an agent's expected return $J(\boldsymbol{\theta})$. The gradient of $J(\boldsymbol{\theta})$ with respect to $\boldsymbol{\theta}$, or *policy gradient*, is often given as:

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

where $A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})$ is the *advantage* of choosing action $\boldsymbol{a}$ in state $\boldsymbol{s}$, quantifying the extra expected reward for choosing $\boldsymbol{a}$ instead of sampling an action from $\pi_{\boldsymbol{\theta}}$. In practice, the expectation in Eq. 1 is approximated with Monte Carlo samples collected from $\pi_{\boldsymbol{\theta}}$, and an estimate of $A^{\pi_{\boldsymbol{\theta}}}$ used in place of the true advantages (Schulman et al., 2016). Currently, the most successful on-policy algorithm is PPO (Schulman et al., 2017), the algorithm of choice in several success stories (Berner et al., 2019; Akkaya et al., 2019; Vinyals et al., 2019). PPO maximizes the clipped surrogate objective:

$$\mathcal{L}_{\text{PPO}} = \min(g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}, \boldsymbol{\theta}_{\text{old}}) A^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{s}, \boldsymbol{a}), \texttt{clip}(g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}, \boldsymbol{\theta}_{\text{old}}), 1 - \epsilon, 1 + \epsilon) A^{\pi_{\boldsymbol{\theta}_{\text{old}}}}(\boldsymbol{s}, \boldsymbol{a})), \tag{2}$$

where $\boldsymbol{\theta}_{\text{old}}$ denotes the policy parameters prior to the update, $g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}, \boldsymbol{\theta}_{\text{old}}) = {}^{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}/_{\pi_{\boldsymbol{\theta}_{\text{old}}}(\boldsymbol{a}|\boldsymbol{s})}$, and the clip function with hyperparameter ϵ clips $g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\theta}, \boldsymbol{\theta}_{\text{old}})$ to the interval $[1 - \epsilon, 1 + \epsilon]$. This objective disincentivizes large changes to $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$. While other policy gradient algorithms perform a single gradient update per data sample to avoid destructively large policy updates, PPO's clipping mechanism permits multiple epochs of minibatch policy updates.

4 Correcting Sampling Error in Reinforcement Learning

In this section, we illustrate how sampling error can produce inaccurate policy gradient estimates and then describe how adaptive, off-policy sampling can reduce sampling error. For exposition, we assume finite state and action spaces. The policy gradient can then be written as:

$$\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) = \sum\nolimits_{(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}} d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) \left[A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}) \right]. \tag{3}$$

The policy gradient is thus a linear combination of the gradient for each $(\boldsymbol{s}, \boldsymbol{a})$ pair $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ weighted by $d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})$. Let $\mathcal{D}$ be a dataset of trajectories. It is straightforward to show that the Monte Carlo estimate of the policy gradient can be written in a similar form as Equation 3 except with the true state-action visitation distribution replaced with the empirical visitation distribution $d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a})$, denoting the fraction of times $(\boldsymbol{s}, \boldsymbol{a})$ appears in $\mathcal{D}$ (Hanna et al., 2021). When $(\boldsymbol{s}, \boldsymbol{a})$ is over-sampled (*i.e.*, $d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a}) > d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})$), then $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ contributes more to the overall gradient than it should. Similarly, when $(\boldsymbol{s}, \boldsymbol{a})$ is under-sampled, $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ contributes less than it should. Below, we provide a concrete example illustrating how small amounts of sampling error can cause the wrong actions to be reinforced.

Example: Sampling error can cause incorrect policy updates

Let $\pi_{\boldsymbol{\theta}}$ be a policy for an MDP with two discrete actions $\boldsymbol{a}_0$ and $\boldsymbol{a}_1$, and suppose that in a particular state $\boldsymbol{s}_0$, the advantage of each action w.r.t. $\pi_{\boldsymbol{\theta}}$ is $A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_0, \boldsymbol{a}_0) = 20$ and $A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_0, \boldsymbol{a}_1) = 15$. For simplicity, suppose the policy has a direct parameterization $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_0|\boldsymbol{s}) = \boldsymbol{\theta}_{\boldsymbol{s}}$, $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}_1|\boldsymbol{s}) = 1 - \boldsymbol{\theta}_{\boldsymbol{s}}$ and places equal probability on both actions in $\boldsymbol{s}_0$ ($\boldsymbol{\theta}_{\boldsymbol{s}_0} = 0.5$). Then, we have $\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_0|\boldsymbol{s}_0) = -\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_1|\boldsymbol{s}_0)$ and $d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_0, \boldsymbol{a}_0) = d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}_0, \boldsymbol{a}_1)$ so that the expected gradient increases the probability of sampling $\boldsymbol{a}_0$, the optimal action. With on-policy sampling, after 10 visits to $\boldsymbol{s}_0$, the agent will sample both actions 5 times in expectation. However, if the agent actually observes $\boldsymbol{a}_0$ 4 times and $\boldsymbol{a}_1$ 6 times, a Monte Carlo estimate of the policy gradient then yields

$$4/10 \cdot 20 \cdot \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_0|\boldsymbol{s}_0) + 6/10 \cdot 15 \cdot \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_1|\boldsymbol{s}_0) = -\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}_0|\boldsymbol{s}_0)$$

which *decreases* the probability of sampling the optimal $\boldsymbol{a}_0$ action.

While sampling error in on-policy sampling vanishes with infinite data, we can eliminate sampling error with finite data by adapting the agent's next action based on previously sampled actions. Suppose the agent visits $\boldsymbol{s}_0$ 9 times and samples $\boldsymbol{a}_0$ 4 times and $\boldsymbol{a}_1$ 5 times. Sampling the next action from a distribution that puts probability 1 on $\boldsymbol{a}_0$ will produce an aggregate batch of data that exactly matches the on-policy distribution and thus produces an accurate gradient. This example suggests a heuristic to reduce sampling error: at each state, select the most under-sampled action. Under the assumption that the MDP has a DAG structure, Zhong et al. (2022) showed that in fixed-horizon settings, this strategy yields an empirical state-action distribution that converges to $d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})$ faster than on-policy sampling. We extend this result by removing the restrictive DAG assumption:

Proposition 1. *Assume that data is collected with an adaptive behavior policy that always takes the most under-sampled action in each state s w.r.t. π, i.e, $a \leftarrow \arg\max_{a'}(\pi(a'|s) - \pi_{\mathcal{D}}(a'|s))$, where $\pi_{\mathcal{D}}$ is the empirical policy after m state-action pairs have been collected. Assume that $\mathcal{S}$ and $\mathcal{A}$ are finite and that the Markov chain induced by π is irreducible. Then we have that the empirical state visitation distribution, d_m, converges to the state distribution of π, d_π, with probability* 1*:*

$$\forall s, \lim_{m \to \infty} d_m(s) = d_\pi(s).$$

We prove Proposition 1 in Appendix A. While this heuristic reduces sampling error, it is difficult to implement in practice; the $\arg\max$ in Proposition 1 often has no closed-form solution, and the empirical policy $\pi_{\mathcal{D}}$ can be expensive to compute at every timestep. The following section presents a *scalable* adaptive sampling algorithm that reduces sampling error in on-policy policy gradient RL.

5 Proximal Robust On-Policy Sampling for Policy Gradient Algorithms

We outline a general framework for on-policy learning with an adaptive behavior policy in Algorithm 1. The behavior $\pi_{\boldsymbol{\phi}}$ policy collects a batch of m transitions, adds it to buffer $\mathcal{D}$, and then updates its weights such that the next batch it collects reduces sampling error in $\mathcal{D}$ with respect to the target policy $\pi_{\boldsymbol{\theta}}$ (Lines 6-8). Every n steps (with $n > m$), the agent updates its target policy with data from $\mathcal{D}$ (Line 9). We refer to m and n as the *behavior batch size* and the *target batch size*, respectively. A subtle implication of adaptive sampling is that it can correct sampling error in *any* empirical data distribution—even one generated by a different policy. Thus, rather than discarding

Algorithm 1 On-policy policy gradient algorithm with adaptive sampling

1: **Inputs**: Target batch size n, behavior batch size m, buffer size b.
2: Initialize target policy parameters $\boldsymbol{\theta}$.
3: Initialize behavior policy parameters $\boldsymbol{\phi} \leftarrow \boldsymbol{\theta}$.
4: Initialize empty buffer $\mathcal{D}$ with capacity bn.
5: **for** target update $i = 1, 2, \ldots$ **do**
6: **for** behavior update $j = 1, \ldots, \lfloor n/m \rfloor$ **do**
7: Collect m samples with $\pi_{\boldsymbol{\phi}}$ and add to $\mathcal{D}$.
8: Update $\pi_{\boldsymbol{\phi}}$ with $\mathcal{D}$ using Algorithm 2.
9: Update $\pi_{\boldsymbol{\theta}}$ with $\mathcal{D}$.

off-policy data from old policies, we let the data buffer hold up to b target batches (bn transitions) and call b the *buffer size*. The behavior policy must continually adjust action probabilities so that the aggregate data distribution of $\mathcal{D}$ matches the expected on-policy distribution of the current target policy (Line 8). Implementing Line 8 is the core challenge we now discuss.

To reduce sampling error, updates to $\pi_{\boldsymbol{\phi}}$ should increase the probability of actions that are under-sampled with respect to $\pi_{\boldsymbol{\theta}}$. Zhong et al. (2022) recently developed Robust On-policy Sampling (ROS) to make such updates. ROS increases the probability of under-sampled actions by updating $\boldsymbol{\phi}$ with a single gradient step in direction $\nabla_{\boldsymbol{\phi}}\mathcal{L} := -\nabla_{\boldsymbol{\phi}} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \log \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})|_{\boldsymbol{\theta}=\boldsymbol{\phi}}$ at each timestep.[1] In theory and in low-dimensional policy evaluation tasks, ROS reduced sampling error at a faster rate compared to on-policy sampling—even when $\mathcal{D}$ contained off-policy data. Unfortunately, two challenges render ROS unsuitable for Line 8 in Algorithm 1.

Challenge 1: Historic data in $\mathcal{D}$ may be very off-policy w.r.t. the current target policy. Since $\nabla_{\boldsymbol{\phi}} \log \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})$ increases in magnitude as $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})$ approaches zero, those off-policy samples can produce destructively large updates.

Challenge 2: In continuous-action tasks, ROS may *increase* sampling error. Continuous policies are typically parameterized as Gaussians with mean $\boldsymbol{\mu}(\boldsymbol{s})$ and diagonal covariance matrix $\Sigma(\boldsymbol{s})$. Since actions in the tails of the Gaussian will generally be under-sampled, the ROS update will continually push the entries of $\boldsymbol{\mu}(\boldsymbol{s})$ toward $\pm\infty$ and the diagonal entries of $\Sigma(\boldsymbol{s})$ toward 0 to increase their probability. The result is a degenerate behavior policy that is too far from the target policy to correct sampling error. We illustrate this scenario in Fig. 6 of Appendix C.

Algorithm 2 PROPS Update

1: **Inputs:** Target policy parameters $\boldsymbol{\theta}$, buffer $\mathcal{D}$, target KL δ_{PROPS}, clipping coefficient ϵ_{PROPS}, regularizer coefficient λ, n_epoch, n_minibatch.
2: $\boldsymbol{\phi} \leftarrow \boldsymbol{\theta}$
3: **for** epoch $i = 1, 2, \ldots,$ n_epoch **do**
4: **for** minibatch $j = 1, 2, \ldots,$ n_minibatch **do**
5: Sample minibatch $\mathcal{D}_j \sim \mathcal{D}$
6: Update $\boldsymbol{\phi}$ with a step of gradient ascent on loss
$$\frac{1}{|\mathcal{D}_j|} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}_j} \mathcal{L}_{\text{PROPS}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}, \epsilon_{\text{PROPS}}, \lambda)$$
7: **if** $D_{\text{KL}}(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\phi}}) > \delta_{\text{PROPS}}$ **then**
8: **break**

We address these challenges with a new behavior policy update. To address Challenge 1, first observe that the gradient of the ROS loss $-\nabla_{\boldsymbol{\phi}}\mathcal{L} = \nabla_{\boldsymbol{\phi}} \log \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})|_{\boldsymbol{\phi}=\boldsymbol{\theta}}$ is equivalent to the policy gradient (Eq. 1) with $A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) = -1, \forall(\boldsymbol{s}, \boldsymbol{a})$. Since the clipped surrogate objective of PPO (Eq. 2) prevents destructively large updates in on-policy policy gradient learning, we can use a similar clipped objective in place of the ROS objective to prevent destructive behavior policy updates:

$$\mathcal{L}_{\text{CLIP}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}, \epsilon_{\text{PROPS}}) = \min\left[-\frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}, -\texttt{clip}\left(\frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}, 1-\epsilon_{\text{PROPS}}, 1+\epsilon_{\text{PROPS}}\right)\right]. \quad (4)$$

Intuitively, this objective is equivalent to the PPO objective (Eq. 2) with $A(\boldsymbol{s}, \boldsymbol{a}) = -1, \forall(\boldsymbol{s}, \boldsymbol{a})$ and incentivizes the agent to decrease the probability of observed actions by at most a factor of $1 - \epsilon_{\text{PROPS}}$. As in PPO, clipping avoids destructively large policy updates and permits us to perform multiple epochs of minibatch updates with the same batch of data. We formally characterize $\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\text{CLIP}}$ in Table 1 of Appendix C. To address the second challenge, we introduce an auxiliary loss that incentivizes the agent to minimize the KL divergence between the behavior policy and target policy at states in the observed data. The full PROPS objective is then:

$$\mathcal{L}_{\text{PROPS}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}, \epsilon_{\text{PROPS}}, \lambda) = \mathcal{L}_{\text{CLIP}}(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) - \lambda D_{\text{KL}}(\pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s})||\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s})) \quad (5)$$

where λ is a regularization coefficient quantifying a trade-off between maximizing $\mathcal{L}_{\text{PROPS}}$ and minimizing D_{KL}. Algorithm 2 details the PROPS update. Like ROS, we set the behavior policy parameters

[1] To provide intuition for the ROS update: gradient ascent on the log-likelihood $\mathcal{L}(\boldsymbol{\phi}) = \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \log \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})|_{\boldsymbol{\theta}=\boldsymbol{\phi}}$ adjusts $\boldsymbol{\phi}$ to better match the empirical distribution of $\mathcal{D}$, increasing the probability of over-sampled actions and decreasing that of under-sampled ones. Taking a step in the opposite direction thus increases the probability of under-sampled actions.

ϕ equal to the target policy parameters at the start of each behavior update, and then make a local adjustment to ϕ to increase the probabilities of under-sampled actions. We stop the PROPS update early when $D_{\text{KL}}(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\phi}})$ reaches a chosen threshold δ_{PROPS}. This technique further mitigates large policy updates and is used in popular implementations of PPO (Raffin et al., 2021; Liang et al., 2018). In Appendix B, we provide theoretical intuition for the relationship between PROPS hyperparameters.

6 Experiments

Our experiments focus on continuous MuJoCo tasks (Brockman et al., 2016) and a tabular 5x5 GridWorld task (Fig. 15a of Appendix E.3). GridWorld contains an optimal goal and a suboptimal goal. While the expected policy gradient increases the probability of reaching the optimal goal, sampling error can yield an empirical gradient that increases the probability of reaching the suboptimal goal and cause agent to converge suboptimally. To converge optimally, the agent must have low sampling error. In all figures, solid curves denote averages and shaded regions denote 95% bootstrap confidence intervals.

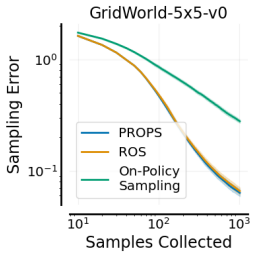
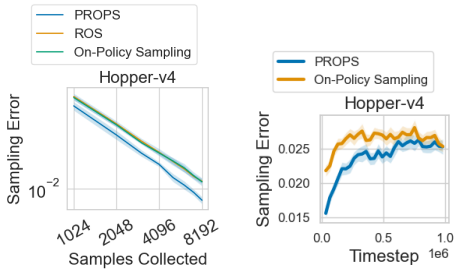


Figure 2: GridWorld sampling error. 10 seeds.

Fixed Target Policy Experiments. We begin by evaluating how quickly PROPS reduces sampling error under a fixed, uniform target policy, similar to the policy evaluation setting considered by Zhong et al. (2022), and compare against two baselines: on-policy sampling and ROS. In GridWorld, we measure sampling error as the total variation distance between the empirical state-action visitation distribution $d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a})$—denoting the proportion of times $(\boldsymbol{s}, \boldsymbol{a})$ appears in $\mathcal{D}$—and the true visitation distribution under $\pi_{\boldsymbol{\theta}}$: $\sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} |d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a}) - d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})|$. In MuJoCo tasks where $d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a})$ is intractable, we follow Zhong et al. (2022) and measure sampling error as the KL divergence $D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}})$ between the empirical policy $\pi_{\mathcal{D}}$ and the target policy $\pi_{\boldsymbol{\theta}}$. We discuss how we compute $\pi_{\mathcal{D}}$ in Appendix D and our hyperparameter search in Appendix E.1.

In GridWorld (Fig. 2), PROPS reduces sampling error faster than on-policy sampling. PROPS and ROS perform similarly, which is expected: with a fixed target policy in a tabular environment, the buffer contains no off-policy data, so Challenges 1 and 2 (from the previous section) do not arise. Appendix E.3 further shows that PROPS yields unbiased and lower-variance estimates than on-policy sampling. In continuous MuJoCo tasks (Fig. 3a), where Challenge 2 occurs, PROPS again reduces sampling error faster than both on-policy sampling and ROS. Notably, ROS offers no improvement over on-policy sampling in all MuJoCo task. This limitation of ROS is unsurprising, as Zhong et al. (2022) showed that ROS struggled to reduce sampling error even in low-dimensional continuous-action tasks. Additional MuJoCo results and ablations of PROPS's clipping and regularization are in Appendix E.1.



(a) Fixed target policy. (b) During RL training.

Figure 3: Sampling error. In (a), the ROS and on-policy sampling curves overlap. 10 seeds

RL Expeiments. We now study PROPS during RL training. Since ROS is computationally expensive and fails to reduce sampling error in MuJoCo tasks even with a fixed policy, we omit it from MuJoCo experiments. We use PPO (Schulman et al., 2017) to update the target policy and consider two baseline methods for providing data to compute PPO updates: vanilla PPO with on-policy sampling, and PPO with on-policy sampling using a buffer of size b (PPO-BUFFER). PPO-BUFFER reuses off-policy data from old policies as if it were on-policy. Although PPO-BUFFER computes biased gradients, it has been successful in prior works (Berner et al., 2019). Since PROPS and PPO-BUFFER have access to the same amount of data for each policy update, any performance difference between these two methods arises from differences in how they sample actions during data collection. In MuJoCo tasks, we use $b = 2$, retaining data for one extra iteration. In GridWorld, we set $b = 1$ and

discard all historic data. To leverage the additional data, we scale the minibatch size for both target and behavior policy updates by a factor of b. We provide hyperparameters in Appendix F.

In GridWorld (Fig. 5a), on-policy sampling achieves a 77% success rate, while PROPS and ROS reach 100%. In MuJoCo tasks (Fig. 4), PROPS achieves higher return than both PPO and PPO-BUFFER throughout training. Figure 12 in Appendix E.2 further shows that the performance profile of PROPS almost always lies above that of PPO and PPO-BUFFER. We now evaluate whether PROPS reduces sampling error more than PPO-BUFFER throughout training. Because PROPS and PPO-BUFFER produce different target policy sequences, we ensure a fair comparison by computing sampling error for on-policy sampling using target policies generated by PROPS. As shown in Fig. 5b, PROPS achieves lower sampling error than PPO-BUFFER in Hopper-v4. We provide ther MuJoCo results in Appendix E.2. In GridWorld, PROPS and ROS reduce sampling error during the first 300 steps and then match on-policy sampling. Since the target policy becomes more deterministic over training, sampling error diminishes, reducing the potential gains from PROPS and ROS. We ablate the clipping coefficient, regularization coefficient, and buffer size in Appendix E.2.

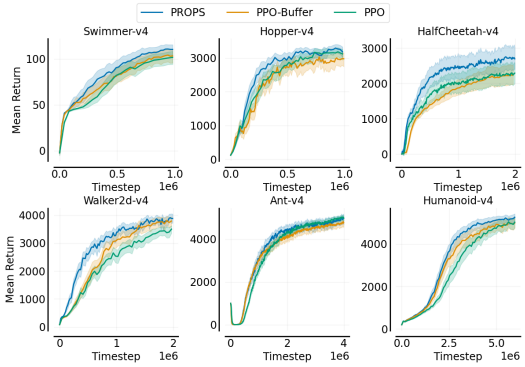
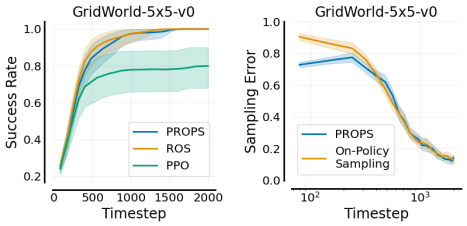


Figure 4: Mean return on MuJoCo tasks. 50 seeds



(a) Success rate. (b) Sampling error.

Figure 5: GridWorld RL. 50 seeds.

7 Limitations and Future Work

PROPS builds on ROS (Zhong et al., 2022), which focused on theoretical analysis and policy evaluation in low-dimensional settings, while we study empirical performance in standard RL benchmarks. A natural next step is to analyze whether PROPS inherits the improved convergence rate shown for ROS. While our results highlight the practical benefits of PROPS, one limitation is that it increases the probability of under-sampled actions regardless of their impact on the gradient. For instance, actions with zero advantage do not influence the gradient and need not be sampled. Future work could prioritize correcting sampling error for $(\boldsymbol{s}, \boldsymbol{a})$ with large $|A^{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})|$. A less obvious feature of PROPS is that it can track the distribution of *any* desired policy, not just the current policy. Thus, PROPS could be integrated into off-policy algorithms to better match a desired exploration policy, enabling more targeted exploration without explicit visitation tracking.

8 Conclusion

In this work, we ask whether on-policy policy gradient methods can learn more efficiently *without* on-policy sampling. To answer this question, we introduce an adaptive, *off-policy* sampling method for on-policy policy gradient RL that collects data such that the empirical distribution of sampled actions closely matches the expected on-policy data distribution at observed states. Our method, Proximal Robust On-policy Sampling (PROPS), periodically updates the data collecting behavior policy to increase the probability of sampling actions that are currently under-sampled with respect to the on-policy distribution. Furthermore, rather than discarding collected data after every policy update, PROPS permits more data efficient on-policy learning by using data collection to adjust the distribution of previously collected data to be approximately on-policy. We replace on-policy sampling with PROPS to collect data for the popular PPO algorithm and empirically demonstrate that PROPS produces data that more closely matches the expected on-policy distribution and yields more data efficient learning compared to on-policy sampling.

9 Acknowledgments

We thank the workshop reviewer for helpful feedback that shaped the final version of this paper. Josiah Hanna is supported in part by the National Science Foundation (IIS-2410981) and Sandia National Labs through a University Partnership Award.

References

Ilge Akkaya, Marcin Andrychowicz, Maciek Chociej, Mateusz Litwin, Bob McGrew, Arthur Petron, Alex Paino, Matthias Plappert, Glenn Powell, Raphael Ribas, et al. Solving rubik's cube with a robot hand. *arXiv preprint arXiv:1910.07113*, 2019.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. Openai gym. *arXiv preprint arXiv:1606.01540*, 2016.

Rasool Fakoor, Pratik Chaudhari, and Alexander J Smola. P3o: Policy-on policy-off policy optimization. In *Uncertainty in Artificial Intelligence*, pp. 1017–1027. PMLR, 2020.

Shixiang Gu, Timothy Lillicrap, Zoubin Ghahramani, Richard E Turner, and Sergey Levine. Q-prop: Sample-efficient policy gradient with an off-policy critic. *arXiv preprint arXiv:1611.02247*, 2016.

Shixiang Shane Gu, Timothy Lillicrap, Richard E Turner, Zoubin Ghahramani, Bernhard Schölkopf, and Sergey Levine. Interpolated policy gradient: Merging on-policy and off-policy gradient estimation for deep reinforcement learning. *Advances in neural information processing systems*, 30, 2017.

Josiah P Hanna, Scott Niekum, and Peter Stone. Importance sampling in reinforcement learning with an estimated behavior policy. *Machine Learning*, 110(6):1267–1317, 2021.

Shengyi Huang, Rousslan Fernand Julien Dossa, Chang Ye, Jeff Braga, Dipam Chakraborty, Kinal Mehta, and João G.M. Araújo. Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms. *Journal of Machine Learning Research*, 23(274):1–18, 2022. URL http://jmlr.org/papers/v23/21-1342.html.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Ksenia Konyushova, Yutian Chen, Thomas Paine, Caglar Gulcehre, Cosmin Paduraru, Daniel J Mankowitz, Misha Denil, and Nando de Freitas. Active offline policy selection. *Advances in Neural Information Processing Systems*, 34:24631–24644, 2021.

Lihong Li, Rémi Munos, and Csaba Szepesvári. Toward minimax off-policy value estimation. In *Artificial Intelligence and Statistics*, pp. 608–616. PMLR, 2015.

Eric Liang, Richard Liaw, Robert Nishihara, Philipp Moritz, Roy Fox, Ken Goldberg, Joseph Gonzalez, Michael Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pp. 3053–3062. PMLR, 2018.

Subhojyoti Mukherjee, Josiah P Hanna, and Robert D Nowak. Revar: Strengthening policy evaluation via reduced variance sampling. In *Uncertainty in Artificial Intelligence*, pp. 1413–1422. PMLR, 2022.

Yusuke Narita, Shota Yasui, and Kohei Yata. Efficient counterfactual learning from bandit feedback. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pp. 4634–4641, 2019.

Brendan O'Donoghue, Rémi Munos, Koray Kavukcuoglu, and Volodymyr Mnih. Pgq: Combining policy gradient and q-learning. *ArXiv*, abs/1611.01626, 2016.

Brahma S. Pavse, Ishan Durugkar, Josiah P. Hanna, and Peter Stone. Reducing sampling error in batch temporal difference learning. In *International Conference on Machine Learning*, 2020.

Doina Precup. Eligibility traces for off-policy policy evaluation. *Computer Science Department Faculty Publication Series*, pp. 80, 2000.

Martin L Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

James Queeney, Ioannis Ch. Paschalidis, and Christos G. Cassandras. Generalized proximal policy optimization with sample reuse. In *Advances in Neural Information Processing Systems*, volume 34. Curran Associates, Inc., 2021.

Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *International Conference on Learning Representations (ICLR)*, 2016.

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

Aaron David Tucker and Thorsten Joachims. Variance-optimal augmentation logging for counterfactual evaluation in contextual bandits. *arXiv preprint arXiv:2202.01721*, 2022.

Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354, 2019.

Runzhe Wan, Branislav Kveton, and Rui Song. Safe exploration for efficient policy evaluation and comparison. In *International Conference on Machine Learning*, pp. 22491–22511. PMLR, 2022.

Ziyu Wang, Victor Bapst, Nicolas Heess, Volodymyr Mnih, Remi Munos, Koray Kavukcuoglu, and Nando de Freitas. Sample efficient actor-critic with experience replay. *arXiv preprint arXiv:1611.01224*, 2016.

Rujie Zhong, Duohan Zhang, Lukas Schäfer, Stefano Albrecht, and Josiah Hanna. Robust on-policy sampling for data-efficient policy evaluation in reinforcement learning. *Advances in Neural Information Processing Systems*, 35:37376–37388, 2022.

A Core Theoretical Results

In this section, we present the proof of Proposition 1. The proof of this result builds upon Lemma 2 by Zhong et al. (2022), which we first restate below for completeness. First, we state an assumption made by Zhong et al. (2022) that is used in their lemma.

Assumption 2. ROS *uses a step-size of* $\alpha \to \infty$ *and the behavior policy is parameterized as a softmax function, i.e.,* $\pi_\theta(a \mid s) \propto e^{\theta_{s,a}}$*, where for each state* s *and action* a*, we have a parameter* $\theta_{s,a}$*. This assumption implies that* ROS *always takes the most under-sampled action in each state.*

Lemma 3. *Let* s *be a state that we visit* m *times. Under ROS sampling, we have* $\forall a \in \mathcal{A}$ *that:*

$$\lim_{m\to\infty} \pi_{\mathcal{D}}(a|s) = \pi(a|s).$$

We now present the proof of Proposition 1. We use d_m, π_m, and p_m as the empirical state visitation distribution, empirical policy, and empirical transition probabilities after m state-action pairs have been taken, respectively. That is, $d_m(s)$ is the proportion of the m states that are s, $\pi_m(a|s)$ is the proportion of the time that action a was observed in state s, and $p_m(s'|s,a)$ is the proportion of the time that the state changed to s' after action a was taken in state s.

Proposition 4. *Assume that data is collected with an adaptive behavior policy that always takes the most under-sampled action in each state,* s*, with respect to policy* π*, i.e,* $a \leftarrow \arg\max_{a'}(\pi(a'|s) - \pi_m(a'|s))$*. We further assume that* $\mathcal{S}$ *and* $\mathcal{A}$ *are finite. Then we have that the empirical state visitation distribution,* d_m*, converges to the state distribution of* π*,* d_π*, with probability* 1*:*

$$\forall s, \lim_{m\to\infty} d_m(s) = d_\pi(s).$$

Proof. The proof of this theorem builds upon Lemma 2 by Zhong et al. (2022) (restated as Lemma 3 above). Note that this lemmas superficially concern the ROS method whereas we are interested in data collection by taking the most under-sampled action at each step. However, as stated in the proof by Zhong et al. (2022), these sampling methods are equivalent under Assumption 2. Thus, we can immediately adopt these lemmas for this proof.

Under Lemma 3, we have that $\lim_{m\to\infty} \pi_m(a|s) = \pi(a|s)$ for any state s under this adaptive data collection procedure. We then have the following $\forall s$:

$$\begin{aligned}
\lim_{m\to\infty} d_m(s) &\overset{(a)}{=} \lim_{m\to\infty} \sum_{\tilde{s}}\sum_{\tilde{a}} p_m(s|\tilde{s},\tilde{a})\pi_m(\tilde{a}|\tilde{s})d_m(\tilde{s}) \\
&= \sum_{\tilde{s}}\sum_{\tilde{a}} \lim_{m\to\infty} p_m(s|\tilde{s},\tilde{a})\pi_m(\tilde{a}|\tilde{s})d_m(\tilde{s}) \\
&= \sum_{\tilde{s}}\sum_{\tilde{a}} \lim_{m\to\infty} p_m(s|\tilde{s},\tilde{a}) \lim_{m\to\infty} \pi_m(\tilde{a}|\tilde{s}) \lim_{m\to\infty} d_m(\tilde{s}) \\
&\overset{(b)}{=} \sum_{\tilde{s}}\sum_{\tilde{a}} p(s|\tilde{s},\tilde{a})\pi(\tilde{a}|\tilde{s}) \lim_{m\to\infty} d_m(\tilde{s}).
\end{aligned}$$

Here, (a) follows from the fact that the empirical frequency of state s can be obtained by considering all possible transitions that lead to s. The last line, (b), holds with probability 1 by the strong law of large numbers and Lemma 3.

We now have a system of $|\mathcal{S}|$ variables and $|\mathcal{S}|$ linear equations. Define variables $x(s) := \lim_{m\to\infty} d_m(s)$ and let $\boldsymbol{x} \in \mathbf{R}^{|\mathcal{S}|}$ be the vector of these variables. We then have $x = P^\pi x$ where $P^\pi \in \mathbf{R}^{|\mathcal{S}|\times|\mathcal{S}|}$ is the transition matrix of the Markov chain induced by running policy π. Assuming that this Markov chain is irreducible, d_π is the unique solution to this system of equations and hence $\lim_{m\to\infty} d_m(s) = d_\pi(s), \forall s$.

□

B Additional Theoretical Results

In this section, we provide additional theory to describe the relationship between different hyperparameters in PROPS:

1. The amount of sampling error in previously collected data and the size of behavior policy updates.
2. The amount of historic data retained by an agent and the amount of additional data the behavior policy must collect to reduce sampling error.

For simplicity, we first focus on a simple bandit setting and then extend to a tabular RL setting.

Suppose we have already collected m state-action pairs and these have been observed with empirical distribution $\pi_m(\boldsymbol{a})$. From what distribution should we sample an additional k state-action pairs so that the empirical distribution over the $m+k$ samples is equal in expectation to $\pi_{\boldsymbol{\theta}}$?

Proposition 5. *Assume that m actions have been collected by running some policy $\pi_{\boldsymbol{\theta}}(\boldsymbol{a})$ and $\pi_m(\boldsymbol{a})$ is the empirical distribution on this dataset. If we collect an additional k state-action pairs using the following distribution, and if $(m+k)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) \geq m \cdot \pi_m(\boldsymbol{a})$, then the aggregate empirical distribution over the $m+k$ pairs is equal to $\pi_{\boldsymbol{\theta}}(\boldsymbol{a})$ in expectation:*

$$\pi_b(\boldsymbol{a}) \coloneqq \frac{1}{Z}\left[\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) + \frac{m}{k}\left(\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})\right)\right]$$

where $Z = \sum_{\boldsymbol{a}\in\mathcal{A}}\left[\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) + \frac{m}{k}\left(\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})\right)\right]$ is a normalization coefficient.

Proof. Observe that $(m+k)\pi_{\boldsymbol{\theta}}(\boldsymbol{a})$ is the expected number of times $\boldsymbol{a}$ is sampled under $\pi_{\boldsymbol{\theta}}$ after $m+k$ steps, $m \cdot \pi_m(\boldsymbol{a})$ is the number of times each $\boldsymbol{a}$ was sampled thus far, and $k \cdot \pi_b(\boldsymbol{a})$ is the expected number of times $\boldsymbol{a}$ is sampled under our behavior policy after k steps. We want to choose $\pi_b(\boldsymbol{a})$ such that $(m+k)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) = m \cdot \pi_m(\boldsymbol{a}) + k \cdot \pi_b(\boldsymbol{a})$ in expectation.

$$\begin{aligned}
(m+k)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) &= k \cdot \pi_b(\boldsymbol{a}) + m \cdot \pi_m(\boldsymbol{a}) \\
-k \cdot \pi_b(\boldsymbol{a}) &= m \cdot \pi_m(\boldsymbol{a}) - (m+k)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) \\
\pi_b(\boldsymbol{a}) &= -\frac{m}{k}\pi_m(\boldsymbol{a}) + \left(\frac{m}{k}+1\right)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) \\
&= \pi_{\boldsymbol{\theta}}(\boldsymbol{a}) + \frac{m}{k}\left(\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})\right)
\end{aligned}$$

Note that $\pi_b(\boldsymbol{a})$ will be a valid probability distribution after normalizing only if

$$\begin{aligned}
\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) + \frac{m}{k}\left(\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})\right) &\geq 0 \\
\left(\frac{m}{k}+1\right)\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) &\geq \frac{m}{k}\pi_m(\boldsymbol{a}) \\
(m+k)\,\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) &\geq m \cdot \pi_m(\boldsymbol{a}).
\end{aligned}$$

If $(m+k)\,\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) < m \cdot \pi_m(\boldsymbol{a})$, then prior to collecting additional data with our behavior policy, $\boldsymbol{a}$ already appears in our data more times in our data than it would in expectation after $m+k$ steps under $\pi_{\boldsymbol{\theta}}$. In other words, we would need to collect more than k additional samples to achieve zero sampling error (or *discard* some previously collected samples).

□

When sampling error is large, behavior policy updates must also be large. Intuitively, the difference $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})$ is the mismatch between the true and empirical visitation distributions, so adding this term to $d_{\pi_{\boldsymbol{\theta}}}$ adjusts $d_{\pi_{\boldsymbol{\theta}}}$ to reduce this mismatch. If $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a}) < 0$, then $\boldsymbol{a}$ is over-sampled w.r.t $\pi_{\boldsymbol{\theta}}$, and π_b will decrease the probability of sampling $\boldsymbol{a}$. If $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a}) > 0$, then $\boldsymbol{a}$ is under-sampled w.r.t $\pi_{\boldsymbol{\theta}}$, and π_b will increase the probability of sampling $\boldsymbol{a}$. When $|\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})|$

is small, the optimal $\pi_b(\boldsymbol{a})$ requires only a small adjustment from $\pi_{\boldsymbol{\theta}}$ (*i.e.*, a small update to the behavior policy is sufficient to reduce sampling error). When $|\pi_{\boldsymbol{\theta}}(\boldsymbol{a}) - \pi_m(\boldsymbol{a})|$ is large, the optimal $\pi_b(\boldsymbol{a})$ requires a large adjustment from $\pi_{\boldsymbol{\theta}}$ (*i.e.*, a large to the behavior policy is needed to reduce sampling error). We can increase (or decrease) the target KL cutoff δ_{PROPS} to allow for larger (or smaller) behavior updates.

When we retain large amounts of historic data, the behavior policy must collect a large amount of additional data to reduce sampling error in the aggregate distribution. The $\frac{m}{k}$ factor implies that how much we adjust $d_{\pi_{\boldsymbol{\theta}}}$ depends on how much data we have already collected (m) and how much additional data we will collect (k). If the k additional samples to collect represent a small fraction of the aggregate $m + k$ samples (*i.e.* $k << m$), then $\frac{m}{k}$ is large, and the adjustment to $d_{\pi_{\boldsymbol{\theta}}}$ is large. This case generally arises when we retain more and more historic data. If the k additional samples to collect represent a large fraction of the aggregate $m + k$ samples (*i.e.* $k >> m$), then $\frac{m}{k}$ is small, and the adjustment to $d_{\pi_{\boldsymbol{\theta}}}$ is small. This case generally arises when we retain little to no historic data.

The next proposition extends this analysis to the tabular RL setting.

Proposition 6. *Assume that m state-action pairs have been collected by running some policy and $d_m(\boldsymbol{s}, \boldsymbol{a})$ is the empirical distribution on this dataset. If we collect an additional k state-action pairs using the following distribution, and if $(m + k)d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) \geq m \cdot d_m(\boldsymbol{s}, \boldsymbol{a})$, then the aggregate empirical distribution over the $m + k$ pairs is equal to $d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})$ in expectation:*

$$d_b(\boldsymbol{s}, \boldsymbol{a}) \coloneqq \frac{1}{Z}\left[d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) + \frac{m}{k}\left(d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) - d_m(\boldsymbol{s}, \boldsymbol{a})\right)\right]$$

where $Z = \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{S}\times\mathcal{A}}\left[d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) + \frac{m}{k}\left(d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) - d_m(\boldsymbol{s}, \boldsymbol{a})\right)\right]$ is a normalization coefficient.

Proof. The proof is identical to the proof of Proposition 5, replacing $\pi_{\boldsymbol{\theta}}(\boldsymbol{a}), \pi_m(\boldsymbol{a})$, and $\pi_b(\boldsymbol{a})$ with $d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}), d_m(\boldsymbol{s}, \boldsymbol{a})$, and $d_b(\boldsymbol{s}, \boldsymbol{a})$. □

In practice, we cannot sample directly from the visitation distribution $d_b(\boldsymbol{s}, \boldsymbol{a})$ in Proposition 6 and instead approximate sampling from this distribution by sampling from its corresponding policy $\pi_b(\boldsymbol{a}|\boldsymbol{s}) = d_b(\boldsymbol{s}, \boldsymbol{a}) / \sum_{(\boldsymbol{s}',\boldsymbol{a}')\in\mathcal{S}\times\mathcal{A}} d_b(\boldsymbol{s}', \boldsymbol{a}')$.

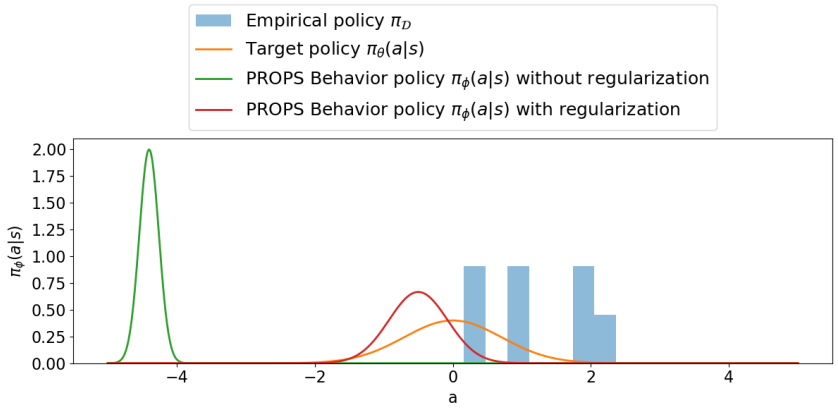


Figure 6: In this example, $\pi(\cdot|\boldsymbol{s}) = \mathcal{N}(0, 1)$. After several visits to $\boldsymbol{s}$, all sampled actions (blue) satisfy $a > 0$ so that actions $a < 0$ are under-sampled. Without regularization, PROPS will attempt to increase the probabilities of under-sampled action in the tail of target policy distribution (green). The regularization term in the PROPS objective ensures the behavior policy remains close to target policy.

$g(\boldsymbol{s}\,\boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) > 0$	Is the objective clipped?	Return value of min	Gradient
$g(\boldsymbol{s}\,\boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) \in [1 - \epsilon_{\text{PROPS}}, 1 + \epsilon_{\text{PROPS}}]$	No	$-g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta})$	$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\text{CLIP}}$
$g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) > 1 + \epsilon_{\text{PROPS}}$	No	$-g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta})$	$\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\text{CLIP}}$
$g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) < 1 - \epsilon_{\text{PROPS}}$	Yes	$-(1 - \epsilon_{\text{PROPS}})$	$\mathbf{0}$

Table 1: Behavior of PROPS's clipped surrogate objective (Eq. 4).

C PROPS Implementation Details

In this appendix, we describe two relevant implementation details for the PROPS update (Algorithm 2) and summarize the behavior of PROPS's clipping mechanism. First, we discuss implementation details.

1. **PROPS update:** The PROPS update adapts the behavior policy to reduce sampling error in the buffer $\mathcal{D}$. When performing this update with a full buffer, we exclude the oldest batch of data collected by the behavior policy (*i.e.*, the m oldest transitions in $\mathcal{D}$); this data will be evicted from the buffer before the next behavior policy update and thus does not contribute to sampling error in $\mathcal{D}$.
2. **Behavior policy class:** We compute behavior policies from the same policy class used for target policies. In particular, we consider Gaussian policies which output a mean $\mu(\boldsymbol{s})$ and a variance $\sigma^2(\boldsymbol{s})$ and then sample actions $\boldsymbol{a} \sim \pi(\cdot|\boldsymbol{s}) \equiv \mathcal{N}(\mu(\boldsymbol{s}), \sigma^2(\boldsymbol{s}))$. In principle, the target and behavior policy classes can be different. However, using the same class for both policies allows us to easily initialize the behavior policy equal to the target policy at the start of each update. This initialization is necessary to ensure the PROPS update increases the probability of sampling actions that are currently under-sampled with respect to the target policy.

We summarize the behavior of PROPS's clipping mechanism in Table 1. Intuitively, the PROPS objective is equivalent to the PPO objective (Eq. 2) with $A(\boldsymbol{s}, \boldsymbol{a}) = -1, \forall(\boldsymbol{s}, \boldsymbol{a})$ and incentivizes the agent to decrease the probability of observed actions by at most a factor of $1 - \epsilon_{\text{PROPS}}$. Let $g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}$. When $g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) < 1 - \epsilon_{\text{PROPS}}$, this objective is clipped at $-(1 - \epsilon_{\text{PROPS}})$. The loss gradient $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\text{CLIP}}$ becomes zero, and the $(\boldsymbol{s}, \boldsymbol{a})$ pair has no effect on the policy update. When $g(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) > 1 - \epsilon_{\text{PROPS}}$, clipping does not apply, and the gradient $\nabla_{\boldsymbol{\phi}} \mathcal{L}_{\text{CLIP}}$ points in a direction that decreases the probability of $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})$.

D Computing Sampling Error

We claim that PROPS improves the data efficiency of on-policy learning by reducing sampling error in the agent's buffer $\mathcal{D}$ with respect to the agent's current (target) policy. To measure sampling error, we use the KL-divergence $D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}})$ between the empirical policy $\pi_{\mathcal{D}}$ and the target policy $\pi_{\boldsymbol{\theta}}$ which is the primary metric Zhong et al. (2022) used to show ROS reduces sampling error:

$$D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\pi_{\mathcal{D}}(\cdot|\boldsymbol{s})}\left[\log\left(\frac{\pi_{\mathcal{D}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}\right)\right]. \tag{6}$$

We compute a parametric estimate of $\pi_{\mathcal{D}}$ by maximizing the log-likelihood of $\mathcal{D}$ over the same policy class used for $\pi_{\boldsymbol{\theta}}$. More concretely, we let $\boldsymbol{\theta}'$ be the parameters of neural network with the same architecture as $\pi_{\boldsymbol{\theta}}$ train and then compute:

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}'} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \log \pi_{\boldsymbol{\theta}'}(\boldsymbol{a}|\boldsymbol{s}) \tag{7}$$

using stochastic gradient ascent. After computing $\boldsymbol{\theta}_{\text{MLE}}$, we then estimate sampling error using the Monte Carlo estimator:

$$D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}}) \approx \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \left(\log \pi_{\boldsymbol{\theta}_{\text{MLE}}}(\boldsymbol{a}|\boldsymbol{s}) - \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})\right). \tag{8}$$

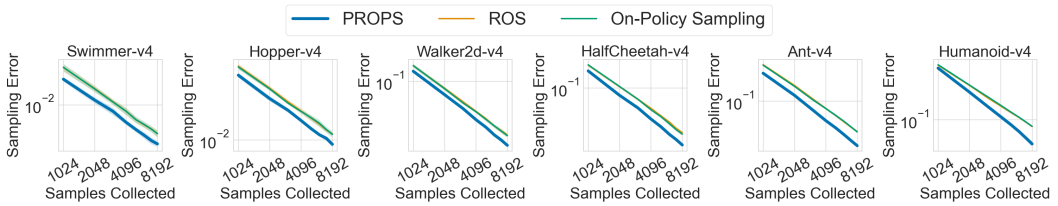


Figure 7: Sampling error with a fixed, randomly initialized target policy. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

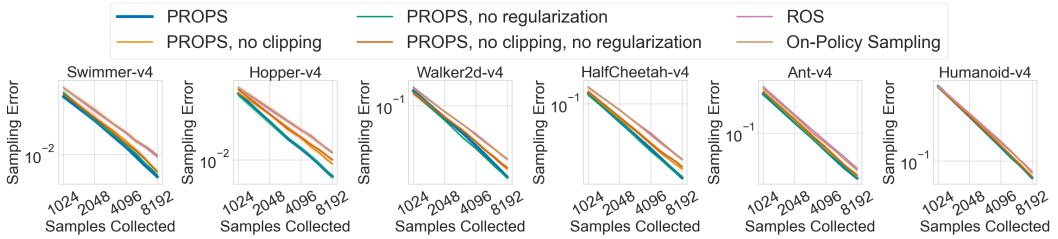


Figure 8: Sampling error ablations with a fixed, expert target policy. Here, "no clipping" refers to setting $\epsilon_{\text{PROPS}} = \infty$, and "no regularization" refers to setting $\lambda = 0$. Solid curves denote the mean over 10 seeds, and shaded regions denote 95% bootstrap confidence intervals.

E Additional Experiments

E.1 Correcting Sampling Error for a Fixed Target Policy

In this appendix, we expand upon results presented in our main experiments and provide additional experiments investigating the degree to which PROPS reduces sampling error with respect to a fixed, randomly initialized target policy. We additionally include ablation studies investigating the effects of clipping and regularization.

We tune PROPS and ROS using a hyperparameter sweep. For PROPS, we sweep over learning rates in $\{10^{-3}, 10^{-4}\}$ and fix the remaining PROPS hyperparameters: regularization coefficient $\lambda = 0.1$, target KL $\delta_{\text{PROPS}} = 0.03$, and clipping coefficient $\epsilon_{\text{PROPS}} = 0.3$. For ROS, we sweep over learning rates in $\{10^{-3}, 10^{-4}, 10^{-5}\}$. We report results for hyperparameters yielding the lowest sampling error.

In Fig. 7, we see that PROPS achieves lower sampling error than both ROS and on-policy sampling across all tasks. ROS shows little to no improvement over on-policy sampling, again highlighting the difficulty of applying ROS to higher dimensional tasks with continuous actions.

Fig. 8 ablates the effects of PROPS's clipping mechanism and regularization on sampling error reduction. We ablate clipping by setting $\epsilon_{\text{PROPS}} = \infty$, and we ablate regularization by setting $\lambda = 0$. We use a fixed expert target policy and use the same tuning procedure described earlier in this appendix. In all tasks, PROPS achieves higher sampling error without clipping nor regularization than it does with clipping and regularization, though this method nevertheless outperforms on-policy sampling in all tasks. Only removing clipping increases sampling error in most setups, and only removing regularization often increases sampling error for smaller batches of data, *e.g.*, 1024 samples. These observations indicate that while regularization in is helpful, clipping has a stronger effect on sampling error reduction than regularization when the target policy is fixed.

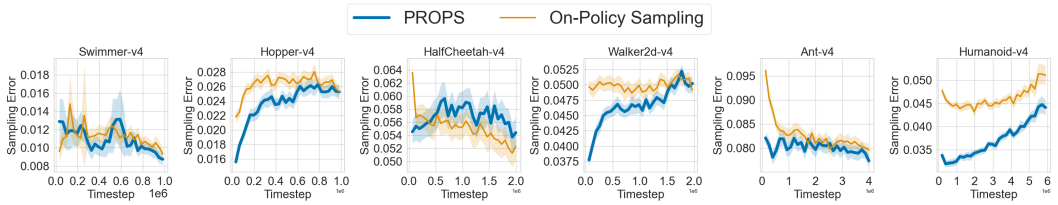


Figure 9: Sampling error throughout RL training. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

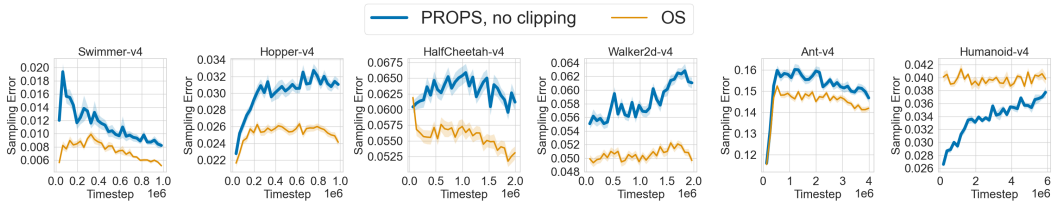


Figure 10: Sampling error throughout RL training without clipping the PROPS objective. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

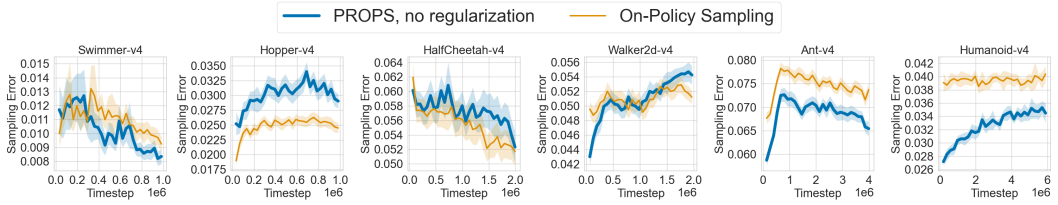


Figure 11: Sampling error throughout RL training without regularizing the PROPS objective. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

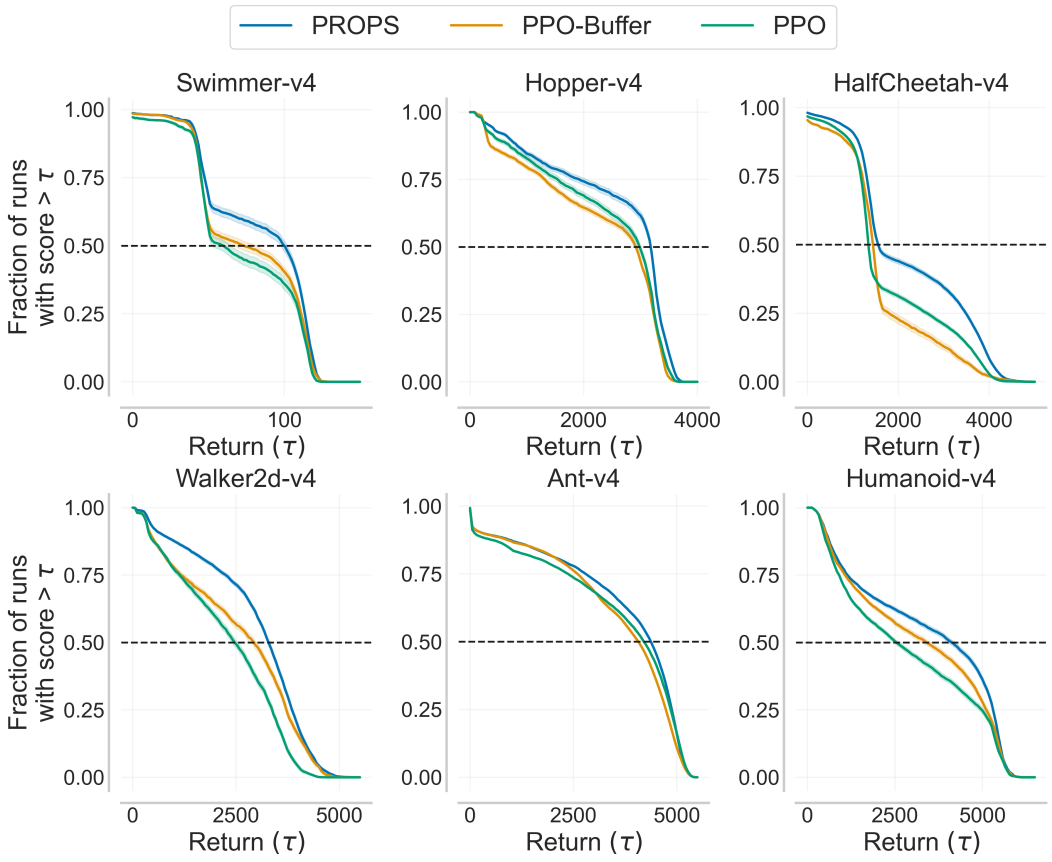


Figure 12: Performance profiles over 50 seeds. Higher values correspond to more reliable convergence to high-return policies. Shaded regions denote 95% bootstrap confidence intervals.

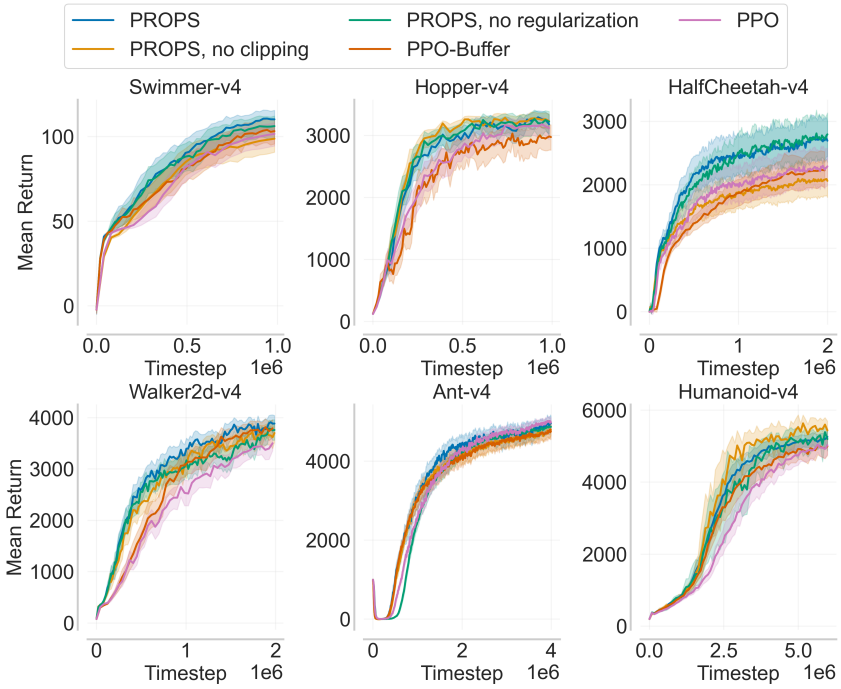


Figure 13: Mean return over 50 seeds of PROPS with and without clipping or regularizing the PROPS objective. Shaded regions denote 95% bootstrap confidence intervals.

E.2 Correcting Sampling Error During RL Training

In this appendix, we include additional experiments investigating the degree to which PROPS reduces sampling error during RL training, expanding upon results presented in Section **??** of the main paper. We include sampling error curves for all six MuJoCo benchmark tasks and additionally provide ablation studies investigating the effects of clipping and regularization on sampling error reduction and data efficiency in the RL setting. We ablate clipping by tuningRL agents with $\epsilon_{\text{PROPS}} = \infty$, and we ablate regularization by tuning RL agents with $\lambda = 0$. Fig. 10 and Fig. 11 show sampling error curves without clipping and without regularization, respectively. Without clipping, PROPS achieves larger sampling than on-policy sampling in all tasks except Humanoid. Without regularization, PROPS achieves larger sampling error in 3 out of 6 tasks. These observations indicate that while clipping and regularization both help reduce sampling during RL training, clipping has a stronger effect on sampling error reduction. As shown in Fig. 13 PROPS data efficiency generally decreases when we remove clipping or regularization.

Lastly, we consider training with larger buffer sizes b in Fig. 14. We find that data efficiency may decrease with a larger buffer size. Intuitively, the more historic data kept around, the more data that must be collected to impact the aggregate data distribution.

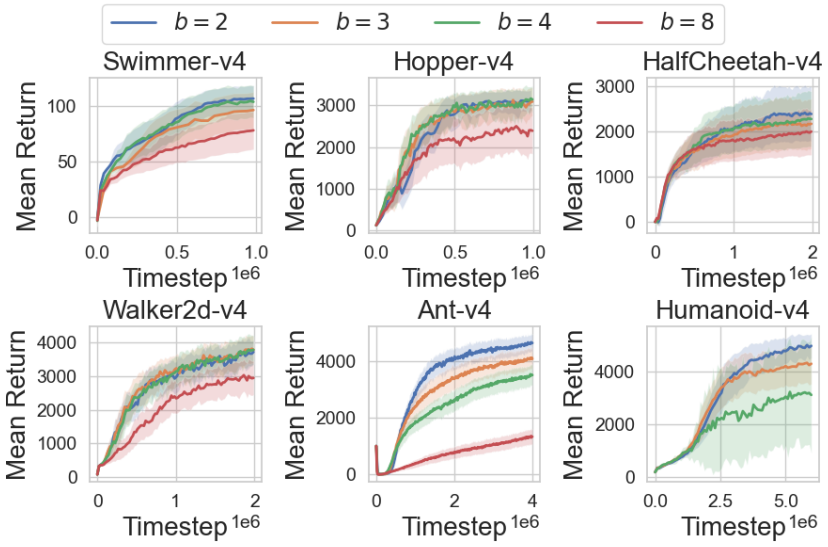
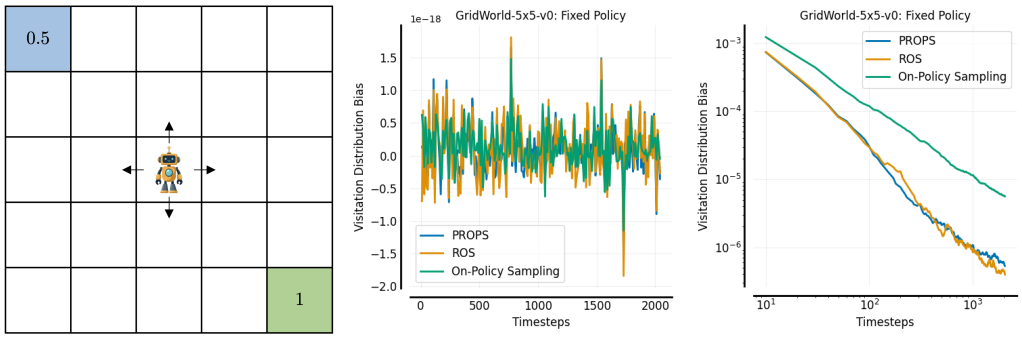


Figure 14: Mean return over 50 seeds for PROPS with different buffer sizes. We exclude $b = 8$ for Humanoid-v4 due to the expense of training and tuning. Shaded regions denote 95% bootstrap confidence intervals.



(a) GridWorld (b) Sampling error bias (c) Sampling error variance

Figure 15: **(a)** GridWorld diagram. The agent starts in the center and receives $+1$ for reaching the bottom right (optimal goal), $+0.5$ for the top left (suboptimal goal), and -0.01 elsewhere. Under an initial uniform policy, the agent is equally likely to observe both goals, so the expected policy gradient points toward an optimal policy. However, with sampling error, the empirical policy gradient can point toward a suboptimal policy that visits the suboptimal goal. **(b, c)** Sampling error bias and variance estimates of different sampling methods. Empirically, PROPS is unbiased and lower variance than on-policy sampling.

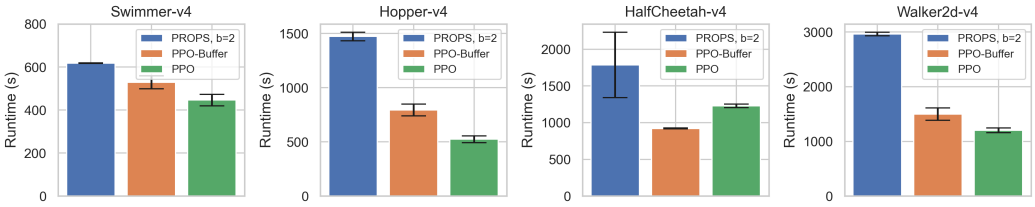


Figure 16: Runtimes for PROPS, PPO-BUFFER, and PPO. We report means and standard errors over 3 independent runs.

E.3 Bias and Variance of PROPS

In Fig. 15, we investigate the bias and variance of the empirical state-action visitation distribution $d^{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a})$ under PROPS, ROS, and on-policy sampling. We report the bias and variance averaged over all $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}$ computed as follows:

$$\text{bias} = \frac{1}{|\mathcal{S} \times \mathcal{A}|} \sum_{(\boldsymbol{s},\boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}} \left(\mathbb{E}\left[d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a}) \right] - d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) \right) \tag{9}$$

$$\text{variance} = \frac{1}{|\mathcal{S} \times \mathcal{A}|} \sum_{(\boldsymbol{s},\boldsymbol{a}) \in \mathcal{S} \times \mathcal{A}} \mathbb{E}\left[\left(d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a}) - d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a}) \right)^2 \right] \tag{10}$$

As shown in Fig. 15, the visitation distribution under PROPS and ROS empirically have near zero bias (note that the vertical axis has scale 10^{-18}) and have lower variance than on-policy sampling.

E.4 Runtime Comparisons

Figure 16 shows runtimes for PROPS, PPO-BUFFER, and PPO averaged over 3 runs. We trained all agents on a MacBook Air with an M1 CPU and use the same tuned hyperparameters used throughout the paper. PROPS takes at most twice as long as PPO-BUFFER; intuitively, both PROPS and PPO-BUFFER learn from the same amount of data but PROPS learns two policies.

PPO learning rate	$10^{-3}, 10^{-4}$, linearly annealed to 0 over training
PPO batch size n	$1024, 2048, 4096, 8192$
PROPS learning rate	$10^{-3}, 10^{-4}$ (and 10^{-5} for Swimmer)
PROPS behavior batch size m	$256, 512, 1024$
PROPS KL cutoff δ_{PROPS}	$0.03, 0.05, 0.1$
PROPS regularizer coefficient λ	$0.1, 0.3$

Table 2: Hyperparameters used in our hyperparameter sweep for RL training.

Environment	Batch Size	Learning Rate
Swimmer-v4	4096	10^{-3}
Hopper-v4	2048	10^{-3}
HalfCheetah-v4	1024	10^{-4}
Walker2d-v4	4096	10^{-4}
Ant-v4	1024	10^{-3}
Humanoid-v4	8192	10^{-4}

Table 3: Tuned PPO hyperparameters

We note that PPO-BUFFER is faster than PPO is HalfCheetah-v4 because, with our tuned hyperparameters, PPO-BUFFER performs fewer target policy updates than PPO. In particular, PPO-BUFFER is updating its target policy every 4096 steps, whereas PPO is updating the target policy every 1024 steps.

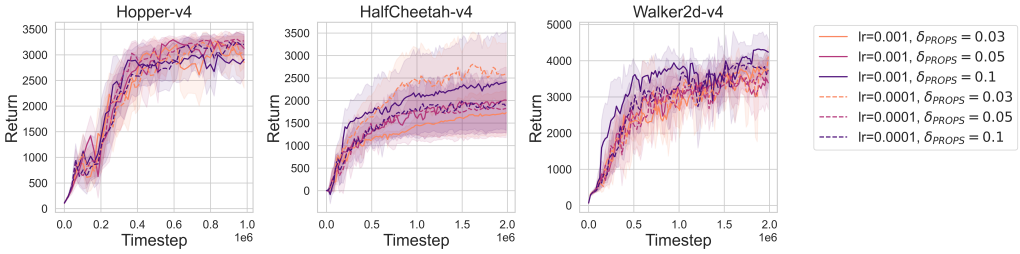
F Hyperparameter Tuning for RL Training

For all RL experiments in Section **??** and Appendix E.2, we tune PROPS, PPO-BUFFER, and PPO separately using a hyperparameter sweep over parameters listed in Table 2 and fix the hyperparameters in Table 5 across all experiments. Since we consider a wide range of hyperparameter values, we ran 10 independent training runs for each hyperparameter setting. We then performed 50 independent training runs for the hyperparameters settings yielding the largest returns at the end of RL training. We report results for these hyperparameters in the main paper. Fig. 17 shows training curves obtained from a subset of our hyperparameter sweep.

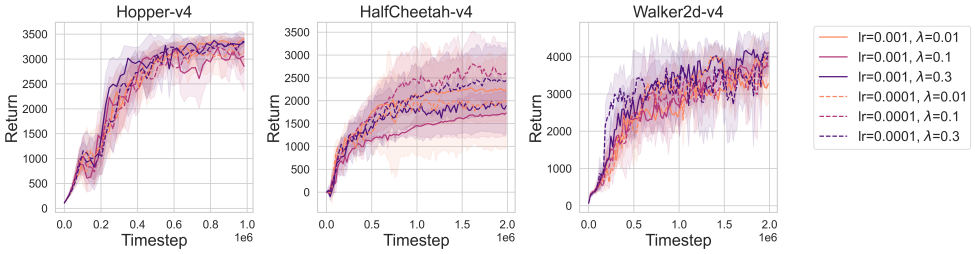
This tuning procedure requires running a few thousand jobs and requires access to many CPUs to do efficiently. We ran all experiments on a compute cluster where we could run several thousand CPU-only jobs in parallel. Our longest jobs (RL training with Humanoid) took at most 12 hours. Jobs for simpler environments like Swimmer finish in under an hour.

Environment	PPO Batch Size	PPO Learning Rate	PROPS Batch Size	PROPS Learning Rate	PROPS KL Cutoff	PROPS Regularization λ
Swimmer-v4	2048	10^{-3}	1024	10^{-5}	0.03	0.1
Hopper-v4	2048	10^{-3}	256	10^{-3}	0.05	0.3
HalfCheetah-v4	1024	10^{-4}	512	10^{-3}	0.05	0.3
Walker2d-v4	2048	10^{-3}	256	10^{-3}	0.1	0.3
Ant-v4	2048	10^{-4}	256	10^{-3}	0.03	0.1
Humanoid-v4	8192	10^{-4}	256	10^{-4}	0.1	0.1

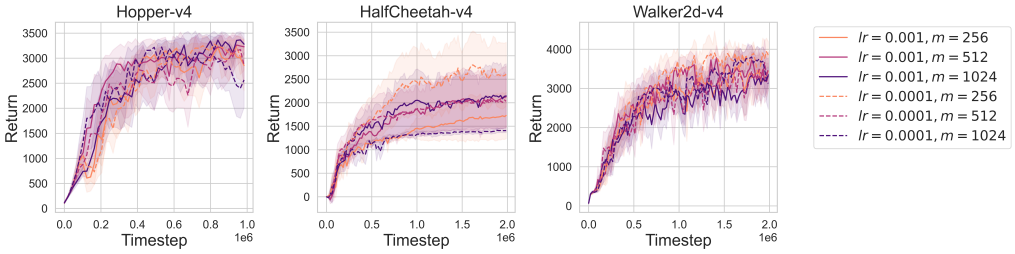
Table 4: Tuned hyperparameters used in RL training with PROPS.



(a) PROPS KL cutoff values δ_{PROPS}.



(b) Regularization coefficients λ.



(c) Behavior batch sizes m (*i.e.* the number of steps between behavior policy updates).

Figure 17: A subset of results obtained from our hyperparameter sweep. Default hyperparameter values are as follows: PROPS KL cutoff $\delta_{\text{PROPS}} = 0.03$; regularization coefficient $\lambda = 0.1$; behavior batch size $m = 256$. Darker colors indicate larger hyperparameter values. Solid and dashed lines have the PROPS learning rate set to $1 \cdot 10^{-3}$ and $1 \cdot 10^{-4}$, respectively. Curves denote averages over 10 seeds, and shaded regions denote 95% confidence intervals.

PPO number of update epochs	10
PROPS number of update epochs	16
Buffer size b	2 target batches (also 3, 4, and 8 in Fig. 14)
PPO minibatch size for PPO update	$bn/16$
PROPS minibatch size for PROPS update	$bn/16$
PPO and PROPS networks	Multi-layer perceptron with hidden layers (64,64)
PPO and PROPS optimizers	Adam (Kingma & Ba, 2015)
PPO discount factor γ	0.99
PPO generalized advantage estimation (GAE)	0.95
PPO advantage normalization	Yes
PPO loss clip coefficient	0.2
PPO entropy coefficient	0.01
PPO value function coefficient	0.5
PPO and PROPS gradient clipping (max gradient norm)	0.5
PPO KL cut-off	0.03
Evaluation frequency	Every 10 target policy updates
Number of evaluation episodes	20

Table 5: Hyperparameters fixed across all experiments. We use the PPO implementation provided by CleanRL (Huang et al., 2022).