$g(\boldsymbol{s}\,\boldsymbol{a}, \boldsymbol{\phi}, \boldsymbol{\theta}) > 0$	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).

A PROPS Implementation Details

In this appendix, we describe two relevant implementation details for the PROPS update (Algorithm 2). We additionally summarize the behavior of PROPS's clipping mechanism in Table 1.

1. **PROPS update:** The PROPS update adapts the behavior policy to reduce sampling error in the replay buffer $\mathcal{D}$. When performing this update with a full replay 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 replay 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.

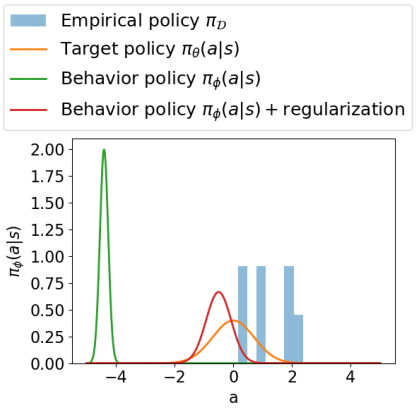


Figure 6: In this example, $\pi(\cdot|\boldsymbol{s}) = \mathcal{N}(0, 1)$. After several visits to $\boldsymbol{s}$, all sampled actions (blue) satisfy $a > 0$ so that actions $a < 0$ are under-sampled. Without regularization, PROPS will attempt to increase the probabilities of under-sampled action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

B Computing Sampling Error

We claim that PROPS improves the data efficiency of on-policy learning by reducing sampling error in the agent's replay 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:

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

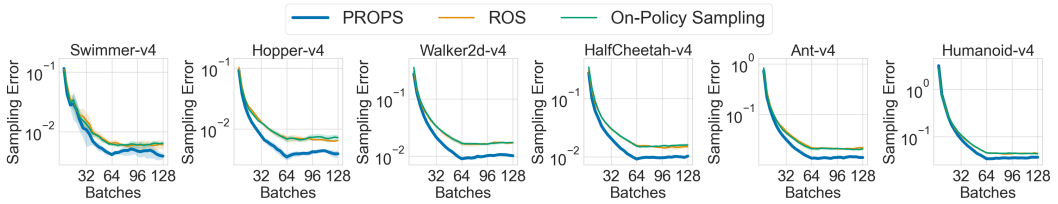


Figure 7: Sampling error with a fixed, expert target policy. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

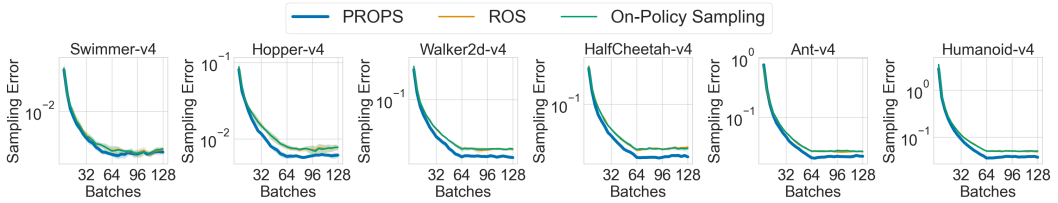


Figure 8: Sampling error with a fixed, randomly initialized target policy. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

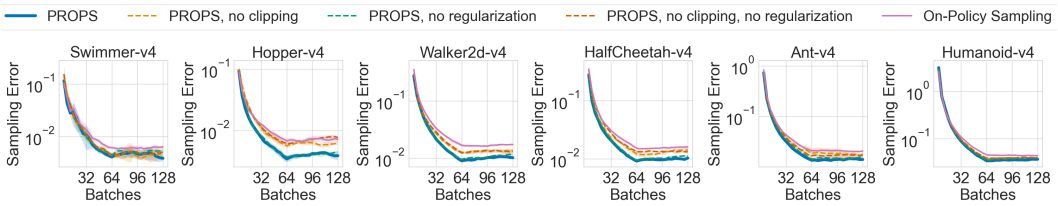


Figure 9: Sampling error ablations with a fixed, expert target policy. Here, "no clipping" refers to setting $\epsilon_{\text{PROPS}} = \infty$, and "no regularization" refers to setting $\lambda = 0$. Solid curves denote the mean over 5 seeds, and shaded regions denote $\pm$ one standard error.

C Correcting Sampling Error for a Fixed Target Policy

In this appendix, we expand upon results presented in Section 6.1 of the main paper and provide additional experiments investigating the degree to which PROPS reduces sampling error with respect to a fixed target policy. We include empirical results for all six MuJoCo benchmark tasks as well as ablation studies investigating the effects of clipping and regularization.

We tune PROPS and ROS using a hyperparameter sweep. For PROPS, we consider learning rates in $\{10^{-3}, 10^{-4}\}$, regularization coefficients $\lambda \in \{0.01, 0.1, 0.3\}$, and PROPS target KLs in $\delta_{\text{PROPS}} \in \{0.05, 0.1\}$. We fix $\epsilon_{\text{PROPS}} = 0.3$ across all experiments. For ROS, we consider learning rates in $\{10^{-3}, 10^{-4}, 10^{-5}\}$. We report results for the hyperparameters yielding the lowest sampling error.

Fig. 7 and 8 show sampling error computed with a fixed expert and randomly initialized target policy, respectively. 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, highlighting the difficulty of applying ROS to higher dimensional tasks with continuous actions.

Fig. 9 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. However, it nevertheless outperforms on-policy sampling in

all tasks except Hopper where it matches the performance of on-policy sampling. Only including regularization slightly decreases sampling error, whereas clipping alone produces sampling error only slightly higher than that achieved by PROPS with both regularization and clipping. 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.

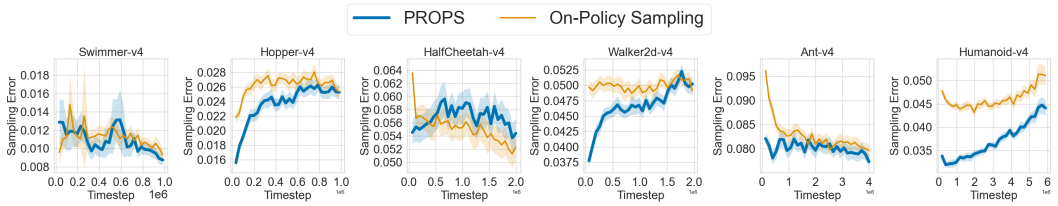


Figure 10: Sampling error throughoutRL training. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

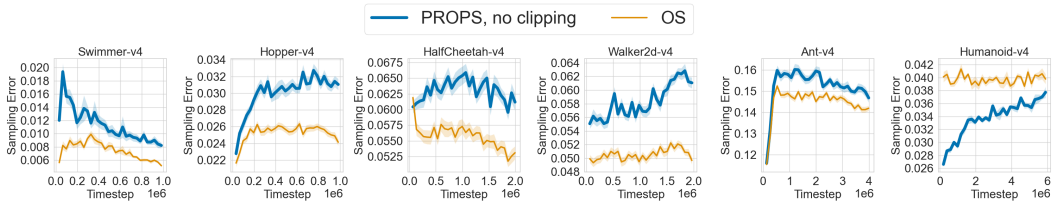


Figure 11: Sampling error throughoutRL training without clipping the PROPS objective. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

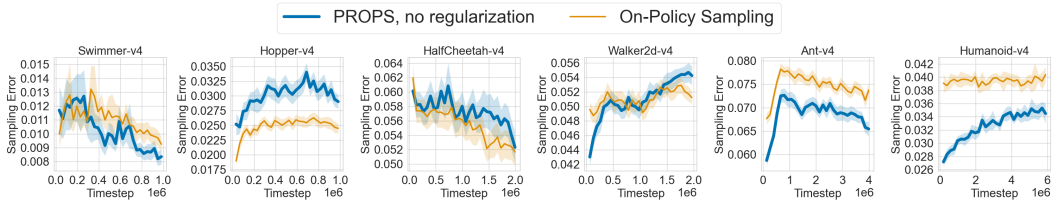


Figure 12: Sampling error throughoutRL training without regularizing the PROPS objective. Solid curves denote the mean over 5 seeds. Shaded regions denote 95% confidence belts.

D 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 6.2 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.

As shown in Fig 10, PROPS achieves lower sampling error than on-policy sampling throughout training in 5 out of 6 tasks. We observe that PROPS increases sampling error but nevertheless improves data efficiency in HalfCheetah as shown in Fig. 5a. This result likely arises from our tuning procedure in which we selected hyperparameters yielding the largest return. Although lower sampling error intuitively correlates with increased data efficiency, it is nevertheless possible to achieve high return without reducing sampling error.

In our next set of experiments, we ablate the effects of PROPS's clipping mechanism and regularization on sampling error reduction and data efficiency. We ablate clipping by tuningRL agents with

$\epsilon_{\text{PROPS}} = \infty$, and we ablate regularization by tuning RL agents with $\lambda = 0$. Fig. 11 and Fig. 12 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.

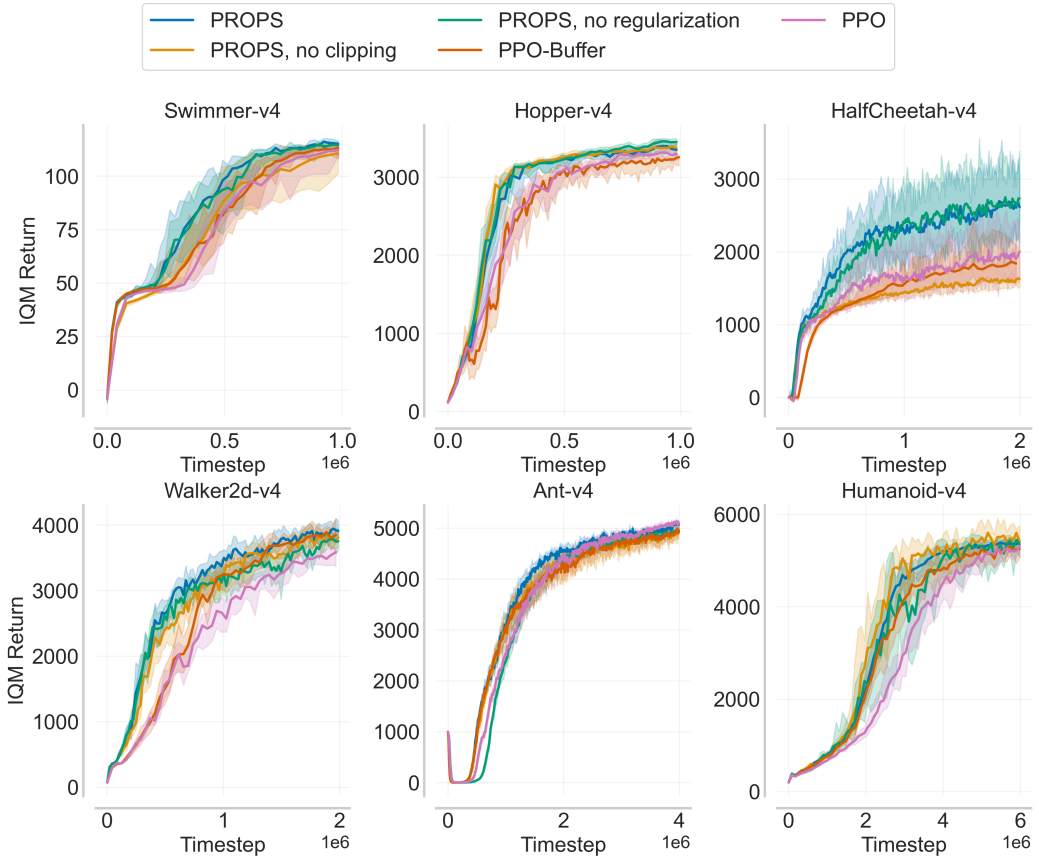


Figure 13: IQM return over 50 seeds of PROPS with and without clipping or regularizing the PROPS objective. Shaded regions denote 95% bootstrapped confidence intervals.

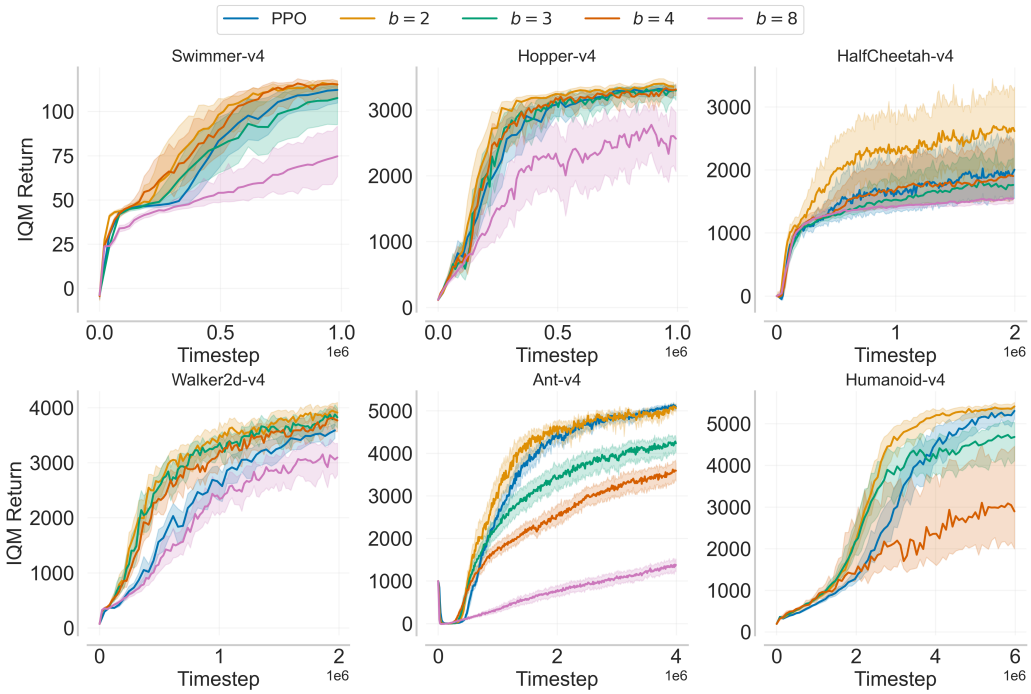


Figure 14: IQM return over 50 seeds for PROPS with different buffer sizes. We exclude $b = 8$ for Humanoid-v4 due to the expense of training and tuning. Shaded regions denote 95% bootstrapped confidence intervals.

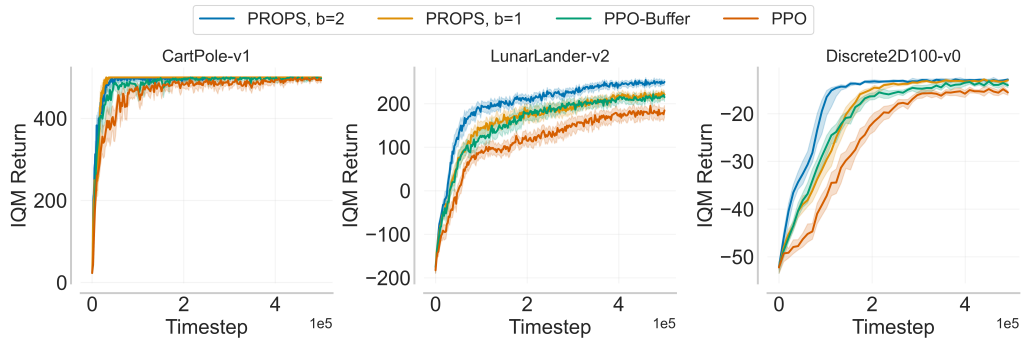


Figure 15: IQM return for discrete action tasks over 50 seeds. Shaded regions denote 95% bootstrapped confidence intervals.

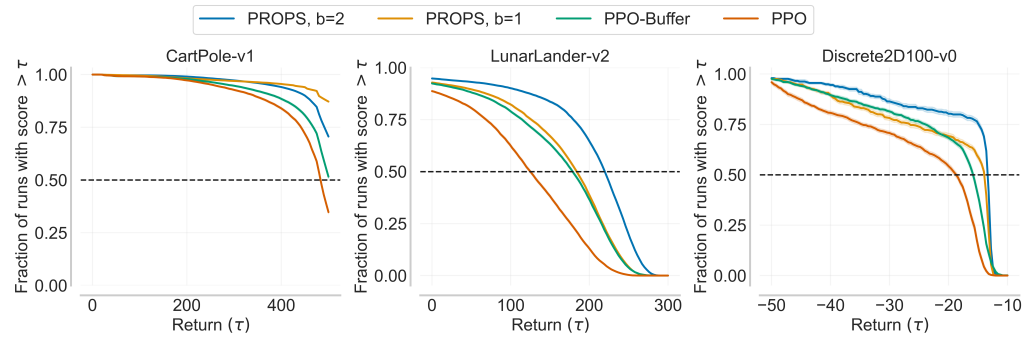


Figure 16: Performance profiles for discrete-action tasks over 50 seeds. The return τ for which the profiles intersect $y = 0.5$ is the median, and the area under the performance profile corresponds to the mean. Shaded regions denote 95% bootstrapped confidence intervals.

E Discrete-Action Tasks

We include 3 additional discrete-action domains of varying complexity. The first two are the widely used OpenAI gym domains CartPole-v1 and LunarLander-v2 (Brockman et al., 2016). The third is a 2D navigation task, Discrete2D100-v0, in which the agent must reach a randomly sampled goal. There are 100 actions, each action corresponding to different directions in which the agent can move. From Fig. 15 and 16 we observe that PROPS with $b = 2$ achieves larger returns than PPO and PPO-BUFFER all throughout training in all three tasks. PROPS with $b = 1$ (no historic data) achieves larger returns than PPO all throughout training in all three tasks and even outperforms PPO-BUFFER in CartPole-v1 and Discrete2D100-v0 even though PPO-BUFFER learns from twice as much data. Thus, PROPS can improve data efficiency *without* historic data.

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	$256, 512, 1024, 2048, 4096$ satisfying $m \leq n$
PROPS KL cutoff δ_{PROPS}	$0.03, 0.05, 0.1$
PROPS regularizer coefficient λ	$0.01, 0.1, 0.3$

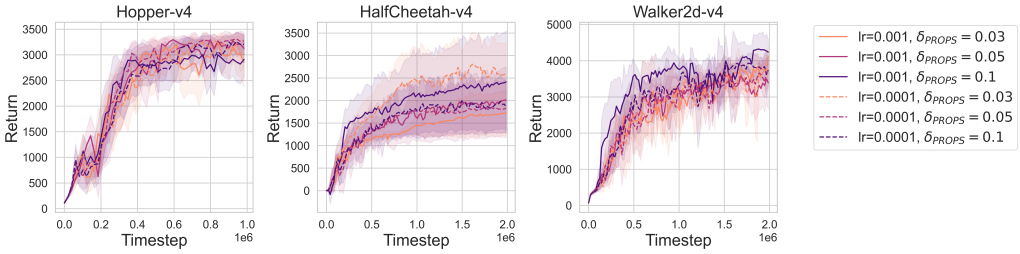
Table 2: Hyperparameters used in our hyperparameter sweep for RL training.

PPO number of update epochs	10
PROPS number of update epochs	16
Replay 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 ROS update	$bn/16$
PPO and PROPS networks	Multi-layer perceptron with hidden layers (64,64)
PPO and PROPS optimizers	Adam (Kingma and 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

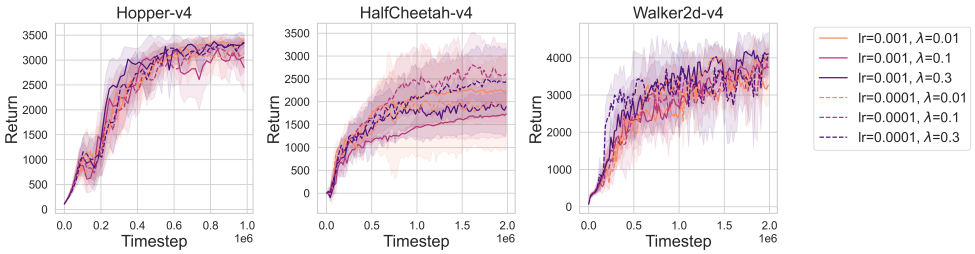
Table 3: Hyperparameters fixed across all experiments. We use the PPO implementation provided by CleanRL (Huang et al., 2022).

F HYPERPARAMETER TUNING FOR RL TRAINING

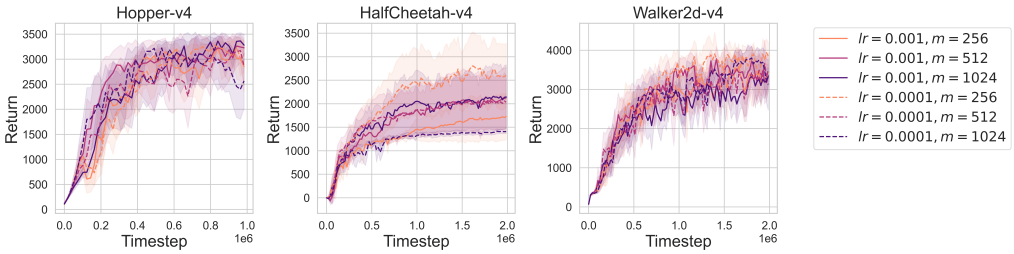
For all RL experiments in Section 6.2 and Appendix D), we tune PROPS, PPO-BUFFER, and PPO separately using a hyperparameter sweep over parameters listed in Table 2 and fix the hyperparameters in Table 3 across all experiments. Since we consider a wide range of hyperparameter values, we ran 10 independent 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.



(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.