Centralized Adaptive Sampling for Reliable Co-Training of Independent Multi-Agent Policies

Nicholas E. Corrado
Department of Computer Sciences
University of Wisconsin–Madison
ncorrado@wisc.edu

Josiah P. Hanna
Department of Computer Sciences
University of Wisconsin–Madison
jphanna@cs.wisc.edu

Abstract

Independent on-policy policy gradient algorithms are widely used for multi-agent reinforcement learning (MARL) in cooperative and no-conflict games, but they are known to converge suboptimally when each agent's policy gradient points toward a suboptimal equilibrium [1, 2, 3, 4]. In this work, we identify a subtler failure mode that arises *even when the expected policy gradients of all agents point toward an optimal solution.* After collecting a finite set of trajectories, stochasticity in independent action sampling can cause the joint data distribution to deviate from the expected joint on-policy distribution. This *sampling error* w.r.t. the joint on-policy distribution produces inaccurate gradient estimates that can lead agents to converge suboptimally. In this paper, we investigate if joint sampling error can be reduced through coordinated action selection and whether doing so improves the reliability of policy gradient learning in MARL. Toward this end, we introduce an adaptive action sampling approach to reduce joint sampling error. Our method, **M**ulti-**A**gent **P**roximal **R**obust **O**n-**P**olicy **S**ampling (MA-PROPS), uses a centralized behavior policy that we continually adapt to place larger probability on joint actions that are currently under-sampled w.r.t. the current joint policy. We empirically evaluate MA-PROPS in a diverse range of multi-agent games and demonstrate that (1) MA-PROPS reduces joint sampling error more efficiently than standard on-policy sampling and (2) improves the reliability of independent policy gradient algorithms, increasing the fraction of training runs that converge to an optimal joint policy.

1 Introduction

On-policy policy gradient methods are among the most popular algorithms used for multi-agent reinforcement learning (RL) in cooperative and no-conflict games [5, 6, 7].[1] A common way to apply these algorithms is to treat agents as independent learners: each agent samples trajectories independently from its own policy (without observing other agents' actions), estimates a Monte Carlo approximation of its policy gradient [8], and performs a local policy update to maximize its expected return via gradient ascent. While independent policy gradient algorithms have demonstrated strong empirical performance across a range of multi-agent benchmarks [3, 9] and powered several high-profile success stories [5, 6, 7], it is well-understood that they may not converge to the most preferred equilibrium even in no-conflict games [1, 2, 3, 4]. In this paper, we identify a subtler, lesser-known failure mode: *even when the expected policy gradients align with optimal behavior*, stochasticity in action sampling can nevertheless cause agents to converge to suboptimal solutions.

To illustrate this issue, consider the 2×2 matrix game in Fig. 1. Suppose each agent assigns equal probability to actions A and B and selects actions independently. For both agents, the expected

[1] In no-conflict games, all agents share the same set of preferred equilibria. Cooperative games are a subset of no-conflict games in which all agents share the same reward function.

Preprint.

reward of A is $(12+0)/2 = 6$, and the expected reward of B is $(6+2)/2 = 4$. Thus, both agents are incentivized to increase the probability of A, steering them toward the optimal joint action (A, A). Now suppose the agents play the game four times. In expectation, each agent will sample both actions twice and observe each joint action once. However, due to randomness in action selection, Agent 1 may sample A, B, A, B while Agent 2 samples B, A, B, A so that (A, A) and (B, B) are *under-sampled* w.r.t. the expected joint on-policy distribution. Consequently, both agents observe reward 0 for action A and reward 6 for action B and thus increase the probability of B, driving agents toward the suboptimal outcome (B, B). The core issue is *joint sampling error*: randomness in action sampling causes the empirical joint data distribution to deviate from the expected joint on-policy distribution, leading to inaccurate policy gradient estimates. Moreover, joint sampling error exists even though both agents sample A and B twice and thus have zero sampling error w.r.t. their own policies.

		Agent 2	
		A	B
Agent 1	A	$12, 12$	$0, 6$
	B	$6, 0$	$2, 2$

Figure 1: 2×2 matrix game where r_1, r_2 denotes rewards for Agent 1 (r_1) and 2 (r_2).

Under on-policy sampling, the only way to reduce sampling error is to collect more data. Recently, Corrado and Hanna [10] introduced an action sampling algorithm (PROPS) that reduces sampling error more efficiently than standard on-policy sampling. However this work focused on reducing sampling error w.r.t. a single policy in single-agent RL settings. As the above example shows, reducing sampling error w.r.t. each agent's policy individually does not necessarily reduce sampling error w.r.t. the joint policy in multi-agent settings. Thus, agents must coordinate their action selection to reduce joint sampling error. These observations motivate the central question of this work: *Can adaptive action sampling reduce joint sampling error and enable independent on-policy policy gradient algorithms to more reliably converge to an optimal joint policy?*

To answer this question, we introduce **M**ulti-**A**gent **P**roximal **R**obust **O**n-**P**olicy **S**ampling (MA-PROPS), an action sampling algorithm that adaptively corrects joint sampling error during multi-agent on-policy data collection. Fig. 2 provides an overview of MA-PROPS. Rather than sampling actions from each agent independently, MA-PROPS samples actions from a separate data collection policy that we continually update to increase the probability of under-sampled joint actions. We evaluate MA-PROPS on a diverse set of multi-agent games and first show that it reduces joint sampling error more efficiently than standard on-policy sampling. Next, we answer our central question affirmatively and show that reducing joint sampling error during independent on-policy policy gradient learning increases the fraction of training runs that converge to an optimal joint policy. While centralized action sampling may be challenging in tasks with many agents due to the exponential growth of the joint action space, we view MA-PROPS as a first step toward understanding and mitigating joint sampling error in multi-agent learning. In summary, our contributions are:

1. We characterize an under-explored failure mode of independent on-policy policy gradient algorithms: even when the expected gradient points toward optimal behavior, joint sampling error can cause convergence to suboptimal solutions.
2. To mitigate this failure mode, we introduce an adaptive sampling method (MA-PROPS) that aims to reduce joint sampling error.
3. We empirically demonstrate that MA-PROPS reduces joint sampling error more efficiently than on-policy sampling and consequently makes independent on-policy policy gradient MARL converge to an optimal joint policy more reliably.

2 Related Work

Reducing sampling error via adaptive data collection. Prior work in single-agent RL has shown that adaptive action sampling can reduce sampling error more efficiently than standard on-policy sampling. Zhong et al. [11] first demonstrated this idea theoretically, and Corrado and Hanna [10] introduced a practical and scalable algorithm (PROPS) for applying it in policy gradient learning. Mukherjee et al. [12] uses adaptive sampling in the bandit setting, and other bandits works [13, 14, 15] use data-conditioned but non-adaptive sampling strategies. These methods focus on single-agent RL or policy evaluation. In contrast, our work focuses on the multi-agent setting and highlights unique challenges posed by joint sampling error.

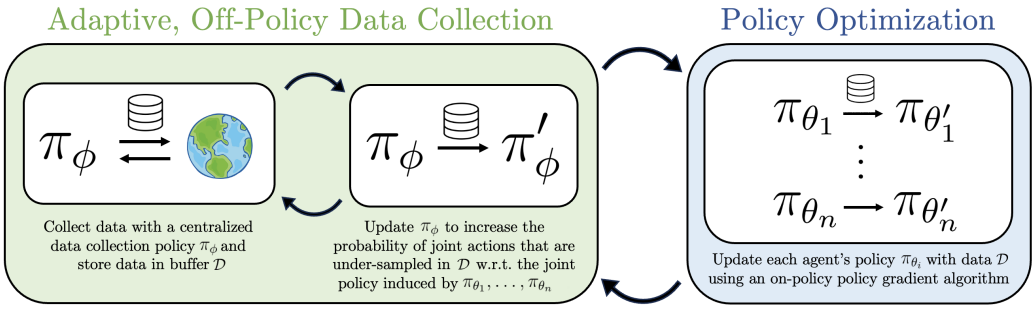


Figure 2: MA-PROPS overview. Rather than collecting data $\mathcal{D}$ by sampling actions from each agent's current policy $\pi_{\boldsymbol{\theta}_i}$, we collect data with a centralized policy $\pi_{\boldsymbol{\phi}}$ that we continually adapt to reduce joint sampling error in $\mathcal{D}$ with respect to the joint policy induced by each agent's policy.

Reducing sampling error via importance sampling. Several works propose importance sampling to reduce sampling error without collecting additional data for policy evaluation [16], policy gradient RL [17, 18, 19], and temporal difference learning [20]. Similar techniques have appeared in the contextual bandit literature [21, 22]. These works reduce sampling error by reweighting previously collected data. In contrast, our work focuses on the question of whether sampling error can be controlled as data is collected.

Equilibrium selection. In games with multiple equilibria, MARL algorithms may converge to suboptimal solutions when each agent's expected gradient points away from the optimal equilibrium. Prior works address equilibrium selection by learning joint action-values [4, 23, 24, 25, 26] or by using optimism to avoid suboptimal equilibria [27, 28, 29]. Most of these works are off-policy methods with the exception of Pareto Actor-Critic [4] and Optimistic Multi-Agent Policy Gradient [29], which modify on-policy gradients to promote convergence to a Pareto-optimal equilibrium. In contrast, we target a different and under-explored failure mode: even when each agent's expected gradient points toward an optimal equilibrium, sampling error in the joint data distribution can produce inaccurate gradient estimates and cause suboptimal convergence. Rather than modifying the gradient update, we improve reliability by reducing joint sampling error through adaptive data collection.

Coordinated Exploration. In MARL, data collection techniques often incentivize agents to maximize state-action coverage and explore states where multi-agent interactions are likely [30, 31, 32, 33, 34, 35, 36]. For instance, MAVEN [30] uses latent variables to promote diverse joint behavior, while influence-based methods [31, 32] guide agents toward states with high mutual information. While these works address exploration, our work focuses on using data collection to more accurately approximate the on-policy distribution and improve on-policy gradient estimates.

3 Preliminaries

3.1 Multi-Agent Reinforcement Learning

We model the MARL environment as a fully observable, finite-horizon stochastic game $(\mathcal{I}, \mathcal{S}, \mathcal{A}, p, r)$ with n agents $\mathcal{I} = \{1, \ldots, n\}$, joint state space $\mathcal{S} = \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$, and joint action space $\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$. Each agent i has a discrete action space $\mathcal{A}_i = \{1, \ldots, k\}$ and a stochastic policy $\pi_{\boldsymbol{\theta}_i} : \mathcal{S}_i \times \mathcal{A}_i \to [0, 1]$ parameterized by $\boldsymbol{\theta}_i$. The joint policy $\pi_{\boldsymbol{\theta}} = (\pi_{\boldsymbol{\theta}_1}, \ldots, \pi_{\boldsymbol{\theta}_n})$ with parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_n)$ defines a distribution over joint actions conditioned on the joint state. For brevity, we often write $\pi_i := \pi_{\boldsymbol{\theta}_i}$ and $\pi := \pi_{\boldsymbol{\theta}}$. The transition function $p : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to [0, 1]$ specifies the probability of transitioning to the next joint state given the current joint state and joint action. The reward function $r : \mathcal{S} \times \mathcal{A} \to \mathbb{R}^n$ returns a reward vector $(r_1, \ldots, r_n)$, where r_i is the reward for agent i. Each agent's expected return is $J_i(\boldsymbol{\theta}) = \mathbb{E}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left[\sum_{t=0}^{H} \gamma^t r_i(\boldsymbol{s}_t, \boldsymbol{a}_t) \right]$, where H is the horizon, and the global objective is to maximize the total return $J(\boldsymbol{\theta}) = \sum_{i \in \mathcal{I}} J_i(\boldsymbol{\theta})$. We focus on *no-conflict games*, where all agents share the same set of optimal joint policies. We refer to the policies used for data collection as *behavior policies* and the policies being optimized as *target policies*.

3.2 On-Policy Policy Gradient Algorithms

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

$$\nabla J_i(\boldsymbol{\theta}) = \mathbb{E}_{\boldsymbol{s}\sim d_\pi, \boldsymbol{a}_i\sim\pi_i, \boldsymbol{a}_{-i}\sim\pi_{-i}} \left[A^\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i}) \nabla \log \pi_i(\boldsymbol{a}_i|\boldsymbol{s})\right]. \tag{1}$$

where d_π denotes the joint state visitation distribution, $\boldsymbol{a}_i$ denotes an action taken by agent i, $\boldsymbol{a}_{-i} = (\boldsymbol{a}_1, \ldots, \boldsymbol{a}_{i-1}, \boldsymbol{a}_{i+1}, \ldots, \boldsymbol{a}_n)$ denotes the actions taken by all agents except agent i, and $A^\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$ is the advantage of choosing joint action $(\boldsymbol{a}_i, \boldsymbol{a}_{-i})$ in state $\boldsymbol{s}$ and following π thereafter. In practice, the expectation in Eq. 1 is approximated with Monte Carlo samples collected from $\pi_1, \ldots, \pi_n$, and an estimate of A^π is used in place of the true advantages [37].

A large body of work extends single-agent policy gradient methods [38, 39, 40, 41, 42, 43, 44] to the multi-agent setting. The simplest of these are independent algorithms like IPPO [5] which apply single-agent algorithms to each agent independently, treating other agents as part of the environment. Since independent updates may converge to suboptimal equilibria [1], many algorithms adopt the Centralized Training with Decentralized Execution (CTDE) paradigm, which gives agents access to global information during training while requiring local execution. CTDE enables straightforward extensions of single-agent policy gradient theorems [45, 46] to multi-agent settings [47], and typically involves using centralized critics or joint advantage estimators to improve credit assignment [48, 47, 6, 49, 50]. Although algorithms like MAPPO [6] are typically not categorized as independent, their centralized value functions depend only on joint observations—not joint actions—so we view them as forms of independent learning.

4 Correcting Sampling Error in MARL

In this section, we formally describe how sampling error w.r.t. the joint on-policy distribution produces inaccurate policy gradient estimates and then describe an adaptive sampling method to reduce joint sampling error. For exposition, we assume finite state and action spaces. The policy gradient w.r.t. agent i can then be written as

$$\nabla J_i(\boldsymbol{\theta}) = \sum\nolimits_{(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i}) \in \mathcal{S}\times\mathcal{A}} d_\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i}) \left[A^\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i}) \nabla_{\boldsymbol{\theta}_i} \log \pi_i(\boldsymbol{a}_i|\boldsymbol{s}_i)\right]. \tag{2}$$

The policy gradient of agent i is thus a linear combination of the gradient for each $(\boldsymbol{s}, \boldsymbol{a}_i)$ pair $\nabla_{\boldsymbol{\theta}_i} \log \pi_i(\boldsymbol{a}_i|\boldsymbol{s})_i$ weighted by $d_\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i}) A^\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$. Crucially, this weighting depends on the *joint* visitation distribution and *joint* advantage. 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 Eq. 2 except with the true state-action visitation distribution replaced with the empirical visitation distribution $d_\mathcal{D}(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$, denoting the fraction of times $(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$ appears in $\mathcal{D}$ [19]. When $(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$ is over-sampled (*i.e.*, $d_\mathcal{D}(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i}) > d_\pi(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$), then $\nabla_{\boldsymbol{\theta}_i} \log \pi_i(\boldsymbol{a}_i|\boldsymbol{s}_i)$ contributes more to the overall gradient than it should. Similarly, when $(\boldsymbol{s}, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$ is under-sampled, $\nabla_{\boldsymbol{\theta}_i} \log \pi_i(\boldsymbol{a}_i|\boldsymbol{s}_i)$ contributes less than it should. In Appendix A, we provide a concrete example based on the matrix game in Fig. 1 illustrating how small amounts of sampling error in the joint data distribution can cause the wrong actions to be reinforced—even when agents have access to the true joint advantages and have zero sampling error in the marginal visitation distribution $d_{\pi_i}(\boldsymbol{s}, \boldsymbol{a}_i)$ of each policy individually.

With on-policy sampling, joint sampling error vanishes in the limit of infinite data. To more rapidly reduce joint sampling error, we can use an alternative sampling strategy: if a joint action is under-sampled at $\boldsymbol{s}$, agents should coordinate to increase the probability of sampling that joint action at $\boldsymbol{s}$ in the future. In tabular *single-agent* settings, Zhong et al. [11] and Corrado and Hanna [10] proved that selecting the most under-sampled action at each state produces an empirical state-action distribution that converges to expected visitation distribution at a faster rate than on-policy sampling. In multi-agent settings, one might try to apply this heuristic to each agent independently. However, as we show in the next example, reducing sampling error w.r.t. each agent individually does not necessarily reduce joint sampling error.

Example: Independent adaptive sampling may not decrease joint sampling error. Consider two policies π_1 and π_2 that assign equal probability to actions A and B so that under on-policy sampling, each joint action is observed equally often in expectation. Now suppose each agent always

selects the action most under-sampled relative to its own policy. At $t = 0$, both actions are equally under-sampled, so both agents sample from π_i. Without loss of generality, suppose agent 1 chooses A and agent 2 chooses B. At $t = 1$, agent 1 selects B and agent 2 selects A, since these actions are now under-sampled. This pattern repeats: the agents choose (A, B) at even timesteps and (B, A) at odd timesteps. Consequently, joint actions (A, A) and (B, B) are never sampled, and joint sampling error does not decrease as $t \to \infty$.

This example highlights a key point: correcting joint sampling error requires agents to coordinate their action selection. Building upon the concepts discussed in this section, we now present a *centralized* adaptive sampling algorithm to correct sampling error w.r.t. joint on-policy distribution.

5 Multi-Agent Robust On-policy Sampling (MA-PROPS)

In this section, we introduce an adaptive sampling algorithm to reduce joint sampling error in multi-agent on-policy data collection. Algorithm 1 outlines our framework, which collects data with a *centralized* behavior policy $\pi_{\boldsymbol{\phi}}$ initialized to match the target joint policy $\pi_{\boldsymbol{\theta}}$. At each step, $\pi_{\boldsymbol{\phi}}$ collects a transition and adds it to buffer $\mathcal{D}$. Every m steps, $\boldsymbol{\phi}$ is updated to increase the probability of under-sampled joint actions in $\mathcal{D}$ w.r.t. $\pi_{\boldsymbol{\theta}}$. Every n steps, each agent updates its policy $\boldsymbol{\theta}_i$ using data from $\mathcal{D}$. To keep the empirical distribution of $\mathcal{D}$ close to the expected joint on-policy distribution, we adapt $\pi_{\boldsymbol{\phi}}$ to place more probability on joint actions that are under-sampled w.r.t. $\pi_{\boldsymbol{\theta}}$. Corrado and Hanna [10] recently introduced an algorithm, PROPS, for making such updates in the single-agent setting. In the remainder of this section, we provide an overview of PROPS and then discuss how to integrate it into the multi-agent setting.

Algorithm 1 On-policy policy gradient algorithm with adaptive sampling

Inputs: Target batch size n, behavior batch size m
Output: Target policy parameters $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n$.
Initialize target policy parameters $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n$.
Initialize behavior policy parameters $\boldsymbol{\phi}$ so $\pi_{\boldsymbol{\phi}} \equiv \pi_{\boldsymbol{\theta}}$.
Initialize empty buffer $\mathcal{D}$.
for timestep $t = 1, 2, \dots$ **do**
  Collect one transition with $\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s})$, add it to $\mathcal{D}$.
  if $t \mod m \equiv 0$ **then**
    Update $\boldsymbol{\phi}$ with $\mathcal{D}$ using Algorithm 2.
  if $t \mod n \equiv 0$ **then**
    Update $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n$ with $\mathcal{D}$ using an on-policy policy gradient algorithm.
return $\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n$

5.1 Behavior Policy Update: Proximal Robust On-Policy Sampling (PROPS)

PROPS is based on a simple idea: starting from $\boldsymbol{\phi} = \boldsymbol{\theta}$, gradient ascent on the log-likelihood $\mathcal{L}_{\text{LL}}(\boldsymbol{\phi}) = \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \log \pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})$ adjusts $\boldsymbol{\phi}$ to 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. Zhong et al. [11] proved that adjusting the behavior policy with a single gradient step in the direction of $-\nabla_{\boldsymbol{\phi}}\mathcal{L}_{\text{LL}}(\boldsymbol{\phi})\big|_{\boldsymbol{\phi}=\boldsymbol{\theta}}$ at each timestep improves the rate at which the empirical data distribution converges to the expected on-policy distribution. To enable larger behavior policy updates that can more aggressively correct sampling error, Corrado and Hanna [10] introduced a PPO-inspired clipped surrogate

$$\mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{s}, \boldsymbol{a}, \varepsilon) = \min\left[-\frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}, -\texttt{clip}\left(\frac{\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}, 1-\varepsilon, 1+\varepsilon\right)\right], \tag{3}$$

where `clip` bounds the ratio $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s})/\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})$ to the interval $[1-\varepsilon, 1+\varepsilon]$. This objective prevents destructively large updates: it can increase the probability of under-sampled actions by at most a factor of $1+\varepsilon$, and decrease that of over-sampled actions by at most $1-\varepsilon$. As in PPO, this enables stable multi-epoch minibatch training while ensuring moderate adjustments to action probabilities.

A natural extension of PROPS to the multi-agent setting uses independent behavior policies $\pi_{\boldsymbol{\phi}_i}(\boldsymbol{a}_i|\boldsymbol{s}_i)$ with the same parameterization as $\pi_{\boldsymbol{\theta}_i}$, yielding a decentralized behavior policy $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s}) = \prod_{i=1}^{n} \pi_{\boldsymbol{\phi}_i}(\boldsymbol{a}_i|\boldsymbol{s}_i)$ where $\boldsymbol{\phi} = (\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_n)$. We provide pseudocode for this approach in Algorithm 2. At each behavior update, we set $\boldsymbol{\phi}_i \leftarrow \boldsymbol{\theta}_i$ and perform minibatch gradient ascent on $\mathcal{L}$. In the next section, we describe how to extend PROPS to directly correct joint sampling error.

Algorithm 2 Behavior policy update with MA-PROPS or PROPS

Inputs: Joint target policy parameters $\boldsymbol{\theta} = (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n)$, buffer $\mathcal{D}$, target KL δ, clipping coefficient ϵ, `n_epoch`, `n_minibatch`.
Output: Behavior policy parameters $\boldsymbol{\phi}$.
if MA-PROPS **then**
  Define *centralized* behavior policy $\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}) := \sigma(\log \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}) + \Delta_{\boldsymbol{\phi}}(\boldsymbol{s}))$
  Initialize $\boldsymbol{\phi}$ such that $\Delta_{\boldsymbol{\phi}}(\boldsymbol{s}) = \mathbf{0}, \forall \boldsymbol{s} \in \mathcal{S}$ so that $\pi_{\boldsymbol{\phi}} \equiv \pi_{\boldsymbol{\theta}}$
else if PROPS **then**
  Define *decentralized* behavior policy $\pi_{\boldsymbol{\phi}}(\boldsymbol{a}|\boldsymbol{s}) = \prod_{i=1}^{n} \pi_{\boldsymbol{\phi}_i}(\boldsymbol{a}_i|\boldsymbol{s}_i)$.
  Initialize $(\boldsymbol{\phi}_1, \dots, \boldsymbol{\phi}_n) \leftarrow (\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_n)$ so that $\pi_{\boldsymbol{\phi}} \equiv \pi_{\boldsymbol{\theta}}$
for epoch $i = 1, 2, \dots,$ `n_epoch` **do**
  for minibatch $j = 1, 2, \dots,$ `n_minibatch` **do**
    Sample minibatch $\mathcal{D}_j \sim \mathcal{D}$
    Update $\boldsymbol{\phi}$ with a step of gradient ascent on loss $\frac{1}{|\mathcal{D}_j|} \sum_{(\boldsymbol{s},\boldsymbol{a}) \in \mathcal{D}_j} \mathcal{L}(\boldsymbol{\phi}, \boldsymbol{\theta}, \boldsymbol{s}, \boldsymbol{a}, \varepsilon)$ (Eq. 3)
  if $D_{\text{KL}}(\pi_{\boldsymbol{\theta}}||\pi_{\boldsymbol{\phi}}) > \delta$ **then**
    return $\boldsymbol{\phi}$
return $\boldsymbol{\phi}$

5.2 Multi-Agent PROPS (MA-PROPS)

To correct sampling error w.r.t. the joint on-policy distribution, the behavior policy $\pi_{\boldsymbol{\phi}}$ must be *centralized*: it must condition on the full joint state $\boldsymbol{s}$ and allow arbitrary dependencies across agents' actions. Since we use PROPS, we also require that $\pi_{\boldsymbol{\phi}}$ can be easily initialized to match $\pi_{\boldsymbol{\theta}}$ at the start of each behavior update. Since $\pi_{\boldsymbol{\phi}}$ and $\pi_{\boldsymbol{\theta}}$ have different parameterizations (*i.e.* $\pi_{\boldsymbol{\theta}}$ is decentralized), we cannot simply set $\boldsymbol{\phi} \leftarrow \boldsymbol{\theta}$. To enable this initialization, we introduce a specialized architecture.

We first compute the logits of the joint policy as $\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}) = \sum_{i=1}^{n} \log \pi_{\boldsymbol{\theta}_i}(\boldsymbol{a}_i|\boldsymbol{s}_i)$. We then define a neural network $\Delta_{\boldsymbol{\phi}}(\boldsymbol{s}) : \mathcal{S} \rightarrow \mathbb{R}^{|\mathcal{A}|}$ that outputs a logit adjustment for each joint action. The behavior policy logits are $\log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s}) + \Delta_{\boldsymbol{\phi}}(\boldsymbol{s})$, and the behavior policy is $\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}) = \sigma\left(\log \pi_{\boldsymbol{\theta}}(\cdot|\boldsymbol{s}) + \Delta_{\boldsymbol{\phi}}(\boldsymbol{s})\right)$, where σ denotes the softmax function.[2] To ensure $\phi_{\boldsymbol{\phi}}$ and $\pi_{\boldsymbol{\theta}}$ are equal at the start of each update, we set the final layer of $\Delta_{\boldsymbol{\phi}}$ to the zero vector so that $\Delta_{\boldsymbol{\phi}}(\boldsymbol{s}) = \mathbf{0}$ for all $\boldsymbol{s} \in \mathcal{S}$. Then, we perform minibatch gradient ascent on $\mathcal{L}(\boldsymbol{\phi})$ to adapt $\Delta_{\boldsymbol{\phi}}$. We call this centralized variant of PROPS Multi-Agent PROPS (MA-PROPS) and provide pseudocode in Algorithm 2.

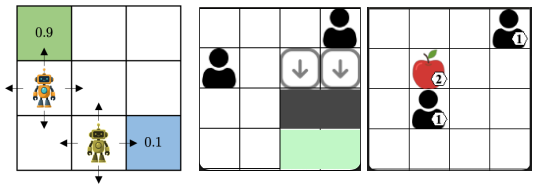
6 Experiments

We design experiments to test the following hypotheses:

- **H1:** MA-PROPS achieves lower sampling error than PROPS and on-policy sampling after collecting a fixed number of samples.
- **H2:** MA-PROPS yields more reliable on-policy policy gradient learning than PROPS and on-policy sampling, increasing the fraction of training runs that converge optimally.

Evaluation metrics. To evaluate **H1**, we use two sampling error metrics. In tasks with small state-action dimensionality, we use the total variation (TV) distance between the empirical joint state-action visitation $d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a})$ distribution, denoting the proportion of times $(\boldsymbol{s}, \boldsymbol{a})$ appears in buffer $\mathcal{D}$, and the true joint state-action visitation distribution $d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})$ under $\pi_{\boldsymbol{\theta}}$: $d_{\text{TV}}(d_{\mathcal{D}}, d_{\pi_{\boldsymbol{\theta}}}) = \frac{1}{2} \sum_{(\boldsymbol{s},\boldsymbol{a}) \in \mathcal{D}} |d_{\mathcal{D}}(\boldsymbol{s}, \boldsymbol{a}) - d_{\pi_{\boldsymbol{\theta}}}(\boldsymbol{s}, \boldsymbol{a})|$. In tasks where it is costly to compute $d_{\pi_{\boldsymbol{\theta}}}$, we follow Corrado and Hanna [10] and Zhong et al. [11] and compute sampling error as the KL divergence $D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}})$, where $\pi_{\mathcal{D}}(\boldsymbol{a}|\boldsymbol{s})$is the empirical policy denoting the fraction of times $\boldsymbol{a}$ was sampled at state $\boldsymbol{s}$ in $\mathcal{D}$. We estimate $\pi_{\mathcal{D}}$ as the maximum likelihood policy under $\mathcal{D}$ (see Appendix B for details). To evaluate **H2**, we periodically evaluate agents over 100 episodes throughout training to track their *success rate*, the fraction of evaluation episodes in which the agents solve the task. In all figures, we plot the mean with 95% bootstrap confidence intervals.

[2]We omit the behavior policy's dependence on $\boldsymbol{\theta}$ for clarity, as $\boldsymbol{\theta}$ is fixed during behavior updates.



(a) GridWorld (b) BoulderPush (c) LBF

		Agent 2		
		A	B	C
Agent 1	A	11	-3	0
	B	-3	7	0
	C	0	3	2

(d) Climbing game.

		Agent 2		
		A	B	C
Agent 1	A	-7	0	10
	B	0	2	0
	C	10	0	-7

(e) Penalty game.

Figure 3: A subset of games used in our experiments. All 2×2 games are listed in Appendix D.

Multi-agent games. We consider Level-Based Foraging (LBF) [9, 51], BoulderPush [4], a 3x3 GridWorld (Fig. 3a), the 3×3 Climbing and Penalty matrix games, and all 21 structurally distinct 2×2 no-conflict matrix games (listed in Fig. 9 of Appendix D) [52]. All tasks require coordination. In GridWorld, agents receive reward 0.9 if they both simultaneously reach the top-left cell (green) and reward -0.1 if only one does. Agents receive reward 0.1 if either agents reaches the bottom-right cell (blue). In BoulderPush, agents must push a boulder to a goal, and both agents receive a penalty if only one agent attempts to push the boulder. Similarly, in LBF, agents must forage a food item together, and both agents receive a penalty if only one agent attempts to forage. We provide additional game details in Appendix D.

Reward rescaling. Our work focuses on improving the reliability of independent on-policy policy gradient algorithms *when the expected policy gradient of each agent points toward optimality at initialization.* However, in most of these games, the expected policy gradient points away from optimality at initialization.[3] The probability of all agents cooperating is very small at initialization, so the expected return of any agent i acting optimally (*e.g.* pushing the boulder) has lower expected return than acting suboptimally (*e.g.* avoiding the boulder) [4, 29]. To ensure games align with our problem of interest, we rescale the reward function in some games so that the expected policy gradient under uniformly initialized policies encourages cooperation. We detail these modifications in Appendix D.

Hyperparameter Tuning. To ensure a fair comparison, we tune MA-PROPS and PROPS over the same hyperparameters and report results for the best setting. See Appendix F for details.

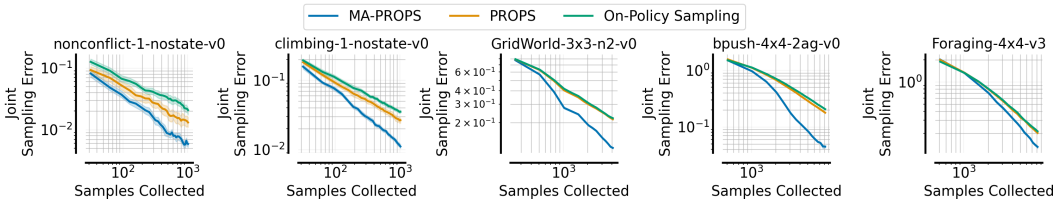
6.1 Reducing Sampling Error w.r.t. a Fixed Target Policy

We first study how quickly MA-PROPS decreases sampling error when each agent's policy is fixed. We provide two baselines for comparison: on-policy sampling and PROPS. We randomly initialize each agent's policy and collect a fixed number of samples using each sampling method. For matrix games and GridWorld, we compute sampling error as $d_{\text{TV}}(d_{\mathcal{D}}, d_{\pi_{\boldsymbol{\theta}}})$. For all other tasks, we use $D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}})$. Since all matrix games have the same dynamics, we focus on 2×2 game 1 and the 3×3 Climbing game. As shown in Fig. 4a, MA-PROPS consistently achieves lower joint sampling error than both PROPS and on-policy sampling. PROPS achieves only marginally lower sampling error compared to on-policy sampling in most tasks. Fig. 4b shows that MA-PROPS achieves the lowest joint sampling error even though PROPS decreases sampling error w.r.t. each agent more than MA-PROPS, highlighting how reducing sampling error w.r.t. each agent does not guarantee reduced joint sampling error. These results support **H1**.

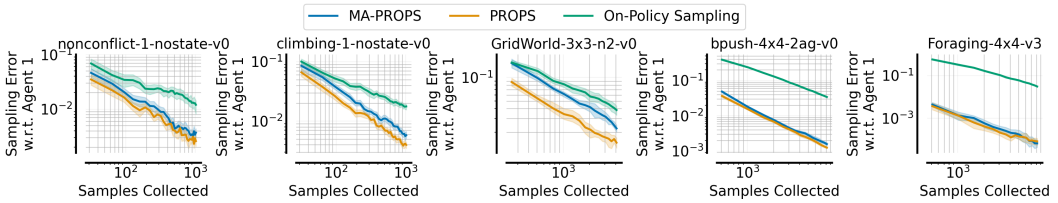
6.2 Reducing Sampling Error During Reinforcement Learning

We now examine how MA-PROPS affects the reliability of on-policy policy gradient algorithms. We train agents using MA-PROPS, PROPS, and on-policy sampling and track their success rate and joint sampling error over training. Our main experiments use MAPPO [6] to update target policies, and we report additional results using IPPO [5] in Appendix E. We compute sampling error as $d_{\text{TV}}(d_{\mathcal{D}}, d_{\pi_{\boldsymbol{\theta}}})$ for matrix games and $D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}})$ for all other tasks. Since MA-PROPS and PROPS generate different target policy sequences during training, we compute on-policy sampling error separately for each method by filling a second buffer with samples from the corresponding target policies.

[3] Christianos et al. [4] show that on-policy policy gradient algorithms like MAPPO and MAA2C consistently converge suboptimally in the same tasks we consider.



(a) Sampling error w.r.t. the joint policy.



(b) Sampling error w.r.t. Agent 1's policy.

Figure 4: Sampling error curves over 10 seeds. **Takeaway: MA-PROPS reduces joint sampling error faster than PROPS and on-policy sampling even though PROPS often reduces sampling error w.r.t. Agent 1 faster than MA-PROPS.**

GridWorld, BoulderPush, LBF. As shown in Fig. 5, MA-PROPS has a higher probability of converging optimally in GridWorld, BoulderPush, and LBF compared to on-policy sampling and PROPS, supporting **H2**. At convergence, MA-PROPS improves success rate over on-policy sampling by 15 percentage points in GridWorld, 19 in BoulderPush, and 10 in LBF. In contrast, PROPS provides only marginal benefit in LBF, no improvement in BoulderPush, and a 20-point drop in GridWorld. Fig. 6 shows joint sampling error during training for BoulderPush and LBF and clarifies why MA-PROPS outperforms PROPS: MA-PROPS decreases joint sampling error more than PROPS. In fact, PROPS does not improve over on-policy sampling at all in LBF. These results support **H1**. We provide joint sampling error curves for GridWorld in Fig. 10 of Appendix E.

Matrix Games. We show training curves for 2×2 games 19-21 and both 3×3 games in Fig. 5. In these games, MA-PROPS is the only algorithm that consistently converges to the optimal policy. We provide training curves for the remaining 2×2 in Appendix E. In games 1–18, PROPS and on-policy sampling achieve success rates of at least 95% of runs. This result is consistent with findings by Christianos et al. [4], who observed that independent on-policy policy gradient algorithms like MAA2C only struggle in games 19–21. Nevertheless, MA-PROPS improves reliability even in these easier games, achieving perfect success rates in all games and converging faster in games 6, 11, 12, 16, and 17. These results support **H2**. We provide all joint sampling error curves for matrix games in Fig. 10 and Fig. 13 of Appendix E. MA-PROPS decreases joint sampling error by a large margin before learning converges, while PROPS often increases joint sampling error. Both MA-PROPS and PROPS reduce sampling error w.r.t. Agent 1, again highlighting how reducing sampling w.r.t. each agent may not reduce joint sampling error. These results support **H1**.

7 Limitations

The core goal of this paper is to characterize a subtle failure mode of independent on-policy policy gradient algorithms—joint sampling error—and then demonstrate that reducing joint sampling via adaptive action sampling can improve the reliability of these algorithms. In this section, we discuss limitations of our proposed approach (MA-PROPS) and outline directions for future work.

Scaling to many agents. The joint action space grows exponentially with the number of agents, making centralized behavior policy learning increasingly difficult. A potential solution is to use deep coordination graphs (DCG) [53] to factor the behavior policy into locally conditioned components, trading off representational capacity for scalability.

Batch size. MA-PROPS increases the probability of under-sampled joint actions at states $\boldsymbol{s}$ in $\mathcal{D}$ so that sampling error is reduced *upon the agents' next visit to* $\boldsymbol{s}$. If $\mathcal{D}$ is small, then revisits are rare,

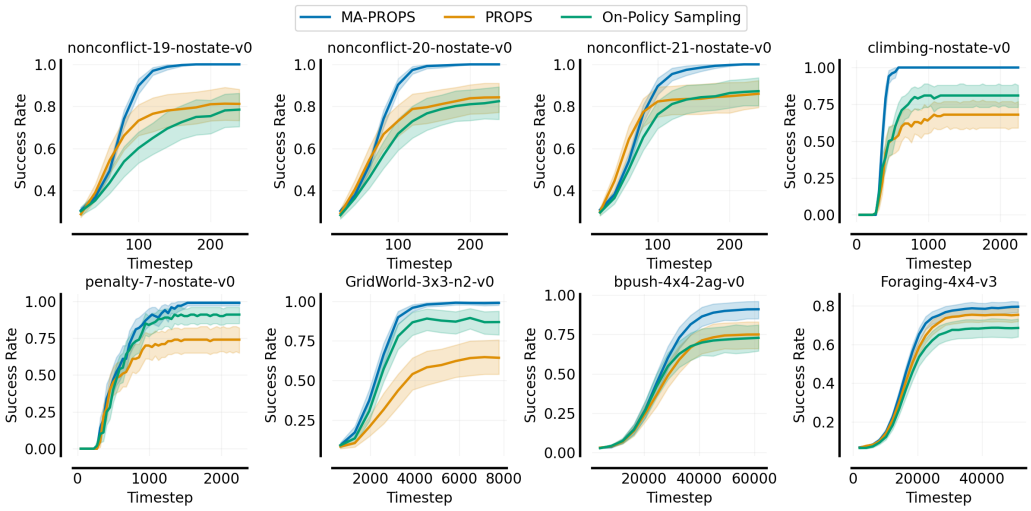
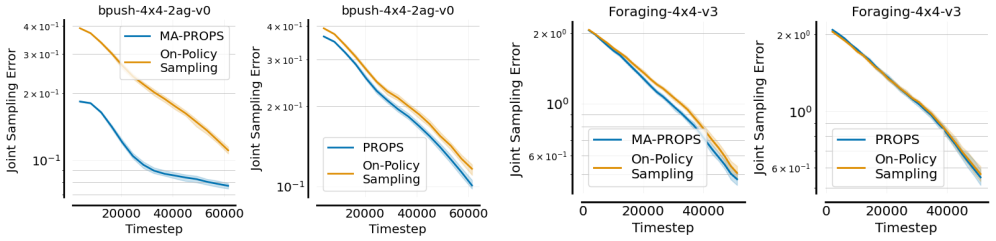


Figure 5: MAPPO training curves over 100 seeds. **Takeaway: MA-PROPS increases success rate 10-20% over on-policy sampling and PROPS. PROPS increases success rate in one task (LBF).**



(a) BoulderPush (b) LBF

Figure 6: Sampling error w.r.t. the joint target policy over 100 seeds. **Takeaway: MA-PROPS reduces joint sampling error more than PROPS.**

and MA-PROPS will perform similarly to on-policy sampling. This could be addressed by letting $\mathcal{D}$ retain historic data between target policy updates. Corrado and Hanna [10] showed that PROPS can leverage historic data by collecting additional data so that the aggregate distribution in $\mathcal{D}$ is close to on-policy. Since MA-PROPS shares the same behavior update rule as PROPS, it can likely benefit from the same data reuse strategies shown to be effective in the single-agent setting.

Discrete Actions. Our empirical evaluation focuses on discrete action tasks, and our centralized behavior policy architecture is specific to discrete-action tasks. However, this architecture easily extends to continuous action settings. Due to space constraints, we detail this extension in Appendix C.

8 Conclusion

In this paper, we identify a subtle failure mode of independent on-policy policy gradient algorithms. Stochasticity in action sampling can cause the joint distribution of collected data to deviate from the expected joint on-policy distribution, and this *sampling error* can cause agents to converge to suboptimal solutions—*even when the expected policy gradients align with optimal behavior*. Given this failure mode, we asked: Can reducing sampling error in the joint data distribution lead to more reliable convergence of independent on-policy policy gradient algorithms? Towards answering this question, we introduce **M**ulti-**A**gent **P**roximal **R**obust **O**n-**P**olicy **S**ampling (MA-PROPS), an action sampling algorithm that adaptively corrects joint sampling error during multi-agent on-policy data collection. MA-PROPS samples actions from a separate data collection policy and periodically updates this policy to increase the probability of under-sampled joint actions. Empirically, MA-PROPS reduces sampling error in the joint on-policy distribution more efficiently than standard on-policy sampling, and increases the fraction of training runs that converge optimally when using independent on-policy policy gradient algorithms.

References

[1] Caroline Claus and Craig Boutilier. The dynamics of reinforcement learning in cooperative multiagent systems. *AAAI/IAAI*, 1998(746-752):2, 1998.

[2] Xueguang Lyu, Yuchen Xiao, Brett Daley, and Christopher Amato. Contrasting centralized and decentralized critics in multi-agent reinforcement learning. *arXiv preprint arXiv:2102.04402*, 2021.

[3] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Comparative evaluation of cooperative multi-agent deep reinforcement learning algorithms. *arXiv preprint arXiv:2006.07869*, 2020.

[4] Filippos Christianos, Georgios Papoudakis, and Stefano V Albrecht. Pareto actor-critic for equilibrium selection in multi-agent reinforcement learning. *arXiv preprint arXiv:2209.14344*, 2022.

[5] Christian Schroeder De Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533*, 2020.

[6] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative multi-agent games. *Advances in neural information processing systems*, 35:24611–24624, 2022.

[7] Ming Zhou, Jun Luo, Julian Villella, Yaodong Yang, David Rusu, Jiayu Miao, Weinan Zhang, Montgomery Alban, Iman Fadakar, Zheng Chen, et al. Smarts: An open-source scalable multi-agent rl training school for autonomous driving. In *Conference on robot learning*, pages 264–285. PMLR, 2021.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. *arXiv preprint arXiv:2006.07869*, 2020.

[10] Nicholas E. Corrado and Josiah P. Hanna. On-policy policy gradient reinforcement learning without on-policy sampling. In *Arxiv Pre-print*, 2023. URL `https://arxiv.org/abs/2311.08290`.

[11] 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.

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

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

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

[15] 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.

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

[17] Matteo Papini, Damiano Binaghi, Giuseppe Canonaco, Matteo Pirotta, and Marcello Restelli. Stochastic variance-reduced policy gradient. In *International conference on machine learning*, pages 4026–4035. PMLR, 2018.

[18] Alberto Maria Metelli, Matteo Papini, Francesco Faccio, and Marcello Restelli. Policy optimization via importance sampling. *Advances in Neural Information Processing Systems*, 31, 2018.

[19] 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.

[20] 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.

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

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

[23] Michael L Littman et al. Friend-or-foe q-learning in general-sum games. In *ICML*, volume 1, pages 322–328, 2001.

[24] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296*, 2017.

[25] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder De Witt, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Monotonic value function factorisation for deep multi-agent reinforcement learning. *Journal of Machine Learning Research*, 21(178):1–51, 2020.

[26] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*, pages 5887–5896. PMLR, 2019.

[27] Laëtitia Matignon, Guillaume J Laurent, and Nadine Le Fort-Piat. Hysteretic q-learning: an algorithm for decentralized reinforcement learning in cooperative multi-agent teams. In *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 64–69. IEEE, 2007.

[28] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani. Lenient multi-agent deep reinforcement learning. *arXiv preprint arXiv:1707.04402*, 2017.

[29] Wenshuai Zhao, Yi Zhao, Zhiyuan Li, Juho Kannala, and Joni Pajarinen. Optimistic multi-agent policy gradient. *arXiv preprint arXiv:2311.01953*, 2023.

[30] Anuj Mahajan, Tabish Rashid, Mikayel Samvelyan, and Shimon Whiteson. Maven: Multi-agent variational exploration. *Advances in neural information processing systems*, 32, 2019.

[31] Tonghan Wang, Jianhao Wang, Yi Wu, and Chongjie Zhang. Influence-based multi-agent exploration. *arXiv preprint arXiv:1910.05512*, 2019.

[32] Iou-Jen Liu, Unnat Jain, Raymond A Yeh, and Alexander Schwing. Cooperative exploration for multi-agent deep reinforcement learning. In *International conference on machine learning*, pages 6826–6836. PMLR, 2021.

[33] Pengyi Li, Hongyao Tang, Tianpei Yang, Xiaotian Hao, Tong Sang, Yan Zheng, Jianye Hao, Matthew E Taylor, Wenyuan Tao, Zhen Wang, et al. Pmic: Improving multi-agent reinforcement learning with progressive mutual information collaboration. *arXiv preprint arXiv:2203.08553*, 2022.

[34] Tarun Gupta, Anuj Mahajan, Bei Peng, Wendelin Böhmer, and Shimon Whiteson. Uneven: Universal value exploration for multi-agent reinforcement learning. In *International Conference on Machine Learning*, pages 3930–3941. PMLR, 2021.

[35] Lulu Zheng, Jiarui Chen, Jianhao Wang, Jiamin He, Yujing Hu, Yingfeng Chen, Changjie Fan, Yang Gao, and Chongjie Zhang. Episodic multi-agent reinforcement learning with curiosity-driven exploration. *Advances in Neural Information Processing Systems*, 34:3757–3769, 2021.

[36] Shaowei Zhang, Jiahan Cao, Lei Yuan, Yang Yu, and De-Chuan Zhan. Self-motivated multi-agent exploration. *arXiv preprint arXiv:2301.02083*, 2023.

[37] 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.

[38] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Reinforcement learning*, pages 5–32, 1992.

[39] Sham M Kakade. A natural policy gradient. *Advances in neural information processing systems*, 14, 2001.

[40] John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR, 2015.

[41] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *International conference on machine learning*, pages 1928–1937. PMLR, 2016.

[42] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Vlad Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, et al. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures. In *International conference on machine learning*, pages 1407–1416. PMLR, 2018.

[43] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[44] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[45] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[46] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr, 2014.

[47] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251*, 2021.

[48] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, volume 32, 2018.

[49] Yanhao Ma and Jie Luo. Value-decomposition multi-agent proximal policy optimization. In *2022 China Automation Congress (CAC)*, pages 3460–3464. IEEE, 2022.

[50] Yifan Zhong, Jakub Grudzien Kuba, Xidong Feng, Siyi Hu, Jiaming Ji, and Yaodong Yang. Heterogeneous-agent reinforcement learning. *Journal of Machine Learning Research*, 25(32): 1–67, 2024.

[51] Filippos Christianos, Lukas Schäfer, and Stefano Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. *Advances in neural information processing systems*, 33: 10707–10717, 2020.

[52] Stefano V Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-agent reinforcement learning: Foundations and modern approaches*. MIT Press, 2024.

[53] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *International Conference on Machine Learning*, pages 980–991. PMLR, 2020.

[54] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *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.

[55] 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.

Appendix

Table of Contents

A Example: Joint Sampling Error Can Cause Incorrect Policy Gradient Updates

In this appendix, we formally illustrate how joint sampling error can lead to inaccurate gradient estimates and consequently suboptimal convergence. For brevity, we write $\nabla := \nabla_{\boldsymbol{\theta}_i}$ to denote the gradient operator w.r.t. agent i's parameters. Consider two policies π_1, π_2 interacting in an MDP with two discrete actions A and B. For simplicity, suppose each policy has a direct parameterization $\pi_i(A|\boldsymbol{s}) = \boldsymbol{\theta}_{i,\boldsymbol{s}}$, $\pi_i(B|\boldsymbol{s}) = 1 - \boldsymbol{\theta}_{i,\boldsymbol{s}}$ with $\boldsymbol{\theta}_{i,\boldsymbol{s}_0} = 0.5$ so that each policy places equal probability on both actions in $\boldsymbol{s}_0$ and thus equal probability on each joint action. Then, we have

$$\begin{aligned} \nabla \log \pi_i(B|\boldsymbol{s}_0) &= -\nabla \log \pi_i(A|\boldsymbol{s}_0), \forall i \\ d_{\pi_i}(\boldsymbol{s}_0, A) &= d_{\pi_i}(\boldsymbol{s}_0, B), \forall i. \end{aligned} \tag{4}$$

Suppose that in a particular state $\boldsymbol{s}_0$ the joint advantages $A^{\pi}(\boldsymbol{s}_0, \boldsymbol{a}_i, \boldsymbol{a}_{-i})$ for both policies are[4]

$$\begin{aligned} A^{\pi}(\boldsymbol{s}_0, A, A) &= 7 \\ A^{\pi}(\boldsymbol{s}_0, A, B) &= -5 \\ A^{\pi}(\boldsymbol{s}_0, B, A) &= 1 \\ A^{\pi}(\boldsymbol{s}_0, B, B) &= -3 \end{aligned}$$

Then, the expected gradient of π_i increases the probability of sampling A and thus increases the probability of observing the optimal joint action (A, A):

$$\frac{2}{4} \cdot (7 - 5) \cdot \nabla \log \pi_i(A|\boldsymbol{s}_0) + \frac{2}{4} \cdot (1 - 3) \cdot \nabla \log \pi_i(B|\boldsymbol{s}_0) = 2\nabla \log \pi_i(A|\boldsymbol{s}_0)$$

With on-policy sampling, after 4 visits to $\boldsymbol{s}_0$, each joint action will be observed once in expectation. However, if we actually observe (A, B) 2 times and (B, A) 2 times, a Monte Carlo estimate of each policy gradient yields

$$\frac{2}{4} \cdot (-5) \cdot \nabla \log \pi_i(A|\boldsymbol{s}_0) + \frac{2}{4} \cdot (-3) \cdot \nabla \log \pi_i(B|\boldsymbol{s}_0) = -\nabla \log \pi_i(A|\boldsymbol{s}_0)$$

which *decreases* the probability of sampling the optimal (A, A) action. We emphasize that this issue arises even though we have zero sampling error w.r.t. the expected on-policy distribution of each policy individually (*i.e.* both agents sample A twice and B twice.)

[4]These are the joint advantages in the 2×2 game from the motivating example in our introduction (Fig. 1).

With on-policy sampling, joint sampling error vanishes in the limit of infinite data. However, this example suggests an alternative sampling strategy to eliminate sampling error with finite data: if a joint action is under-sampled at $\boldsymbol{s}$, agents should coordinate to increase the probability of sampling that joint action at $\boldsymbol{s}$ in the future. Continuing with our example, suppose the agents will visit $\boldsymbol{s}_0$ 4 more times. To achieve zero sampling error, the agents must observe (A, A) and (B, B) twice each. With on-policy sampling, they may observe (A, B) and (B, A) again. If they sample their next actions from a distribution that puts probability 1 on (A, A) on the first two visits to $\boldsymbol{s}_0$ and probability 1 on (B, B) on the next two, the aggregate batch of data will exactly match the expected joint distribution.

B Computing Sampling Error

Similar to Zhong et al. [11] and Corrado and Hanna [10], we measure sampling error as the KL-divergence $D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}})$ between the empirical joint policy $\pi_{\mathcal{D}}$ and the target joint policy $\pi_{\boldsymbol{\theta}}$:

$$D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}}) = \mathbb{E}_{\boldsymbol{s}\sim\mathcal{D},\boldsymbol{a}\sim\pi_{\mathcal{D}}(\cdot|\boldsymbol{s})}\left[\log\left(\frac{\pi_{\mathcal{D}}(\boldsymbol{a}|\boldsymbol{s})}{\pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})}\right)\right]. \tag{5}$$

We compute a parametric estimate of $\pi_{\mathcal{D}}$ by letting $\boldsymbol{\theta}'$ be the parameters of neural network that takes joint states as input and outputs a distribution over joint actions $\pi_{\boldsymbol{\theta}'}(\cdot|\boldsymbol{s})$ and maximizing the log-likelihood of $\mathcal{D}$ under $\pi'_{\boldsymbol{\theta}}$

$$\boldsymbol{\theta}_{\text{MLE}} = \arg\max_{\boldsymbol{\theta}'} \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \log \pi_{\boldsymbol{\theta}'}(\boldsymbol{a}|\boldsymbol{s}) \tag{6}$$

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

$$D_{\text{KL}}(\pi_{\mathcal{D}}||\pi_{\boldsymbol{\theta}}) \approx \sum_{(\boldsymbol{s},\boldsymbol{a})\in\mathcal{D}} \left(\log \pi_{\boldsymbol{\theta}_{\text{MLE}}}(\boldsymbol{a}|\boldsymbol{s}) - \log \pi_{\boldsymbol{\theta}}(\boldsymbol{a}|\boldsymbol{s})\right). \tag{7}$$

We compute sampling error w.r.t. individual agents in a similar fashion: We compute estimate $\pi_{\mathcal{D}_i}$ as the maximum likelihood policy under $\mathcal{D}_i$ and then estimate sampling error using the the Monte Carlo estimator:

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

C Extending MA-PROPS to Continuous Action Tasks

The centralized behavior policy we use is specific to softmax policies for discrete action settings. However, this architecture easily extends to continuous action settings. Continuous policies are typically parameterized as Gaussians $\mathcal{N}(\boldsymbol{\mu}_i(\boldsymbol{s}_i), \sigma_i(\boldsymbol{s}_i))$ with mean $\boldsymbol{\mu}_i(\boldsymbol{s}_i)$ and variance $\sigma_i(\boldsymbol{s}_i)$. The joint policy is then a Gaussian with mean $\boldsymbol{\mu}(\boldsymbol{s}) = (\boldsymbol{\mu}_1(\boldsymbol{s}_1), \ldots, \boldsymbol{\mu}_n(\boldsymbol{s}_n))$ and variance $\sigma(\boldsymbol{s}) = (\sigma_1(\boldsymbol{s}_1), \ldots, \sigma_n(\boldsymbol{s}_n))$. Similar to how the behavior network add an adjustment to the joint logits in the discrete setting, in a continuous setting, the behavior network would output an adjustment to the joint mean and joint variance.

More concretely, we first compute the mean $\boldsymbol{\mu}(\boldsymbol{s})$ and variance $\sigma(\boldsymbol{s})$ of the joint policy. Next, we define a neural network $\Delta_{\boldsymbol{\phi}}(\boldsymbol{s}) : \mathcal{S} \to \mathbb{R}^{|\mathcal{A}|}$ that outputs an adjustment to the mean and variance of each joint action dimension. Let $\Delta^{\mu}_{\boldsymbol{\phi}}(\boldsymbol{s})$ denote the mean adjustments and let $\Delta^{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{s})$ denote the variance adjustments. The behavior policy is then[5]

$$\pi_{\boldsymbol{\phi}}(\cdot|\boldsymbol{s}) = \mathcal{N}\left(\mu(\boldsymbol{s}) + \Delta^{\mu}_{\boldsymbol{\phi}}(\boldsymbol{s}), \sigma(\boldsymbol{s}) + \Delta^{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{s})\right) \tag{9}$$

To initialize behavior policy to match $\pi_{\boldsymbol{\theta}}$ at the start of each update, we set the final layer of $\Delta_{\boldsymbol{\phi}}$ to the zero vector so that $\Delta^{\mu}_{\boldsymbol{\phi}}(\boldsymbol{s}) = \mathbf{0}$ and $\Delta^{\sigma}_{\boldsymbol{\phi}}(\boldsymbol{s}) = \mathbf{0}$ for all $\boldsymbol{s} \in \mathcal{S}$. Then, we perform minibatch gradient ascent on $\mathcal{L}(\boldsymbol{\phi})$ to adapt the logit adjustment so that the behavior policy places larger probability on under-sampled actions just as described in Algorithm 2.

[5]We again omit the behavior policy's dependence on $\boldsymbol{\theta}$ for clarity, as $\boldsymbol{\theta}$ is fixed during behavior updates.

		Agent 2		
		A	B	C
Agent 1	A	11	-30	0
	B	-30	7	0
	C	0	6	5

(a) Original Climbing game [1].

		Agent 2		
		A	B	C
Agent 1	A	11	-3	0
	B	-3	7	0
	C	0	3	2

(b) Modified Climbing game.

		Agent 2		
		A	B	C
Agent 1	A	$-k$	0	10
	B	0	2	0
	C	10	0	$-k$

(a) Original Penalty game [1].

		Agent 2		
		A	B	C
Agent 1	A	-7	0	10
	B	0	2	0
	C	10	0	-7

(b) Modified Penalty game. We choose $k = 7$.

Figure 8: Original and modified 3×3 matrix games.

D Multi-Agent Games

In this appendix, we further describe each multi-agent game we use. We additionally detail how we modify some games to ensure the expected policy gradient encourages cooperation at initialization.

D.1 Game Descriptions

BoulderPush: In this game, agents must coordinate to push a boulder to a specified goal position. Agents have four actions that move them one position up, down, left, or right. To push the boulder, all agents must move to the positions above the boulder (*i.e.*, the cells indicating the direction in which the boulder must be pushed) and then move in the indicated direction. Agents receive reward 0.1 for successfully pushing the boulder. If only one agent attempts to push the boulder independently (without the other agent), they receive reward -0.02.

Level-based Foraging: In this game, agents must coordinate to collect food items on the game board. Agents have five actions; four actions can move them one position up, down, left, or right. The fifth action is the "forage" action where the agent attempts to forage a food item. To forage the food, both agents must move next to it and choose the forage action. Agents receive reward 0.5 for successfully foraging the food. If only one agent attempts to forage independently (without the other agent), they receive reward -0.015.

D.2 Reward Modifications

Our work focuses on improving the reliability of independent on-policy policy gradient algorithms *when the expected policy gradient of each agent points toward optimality at initialization.* However, in most of these games, the expected policy gradient points away from optimality at initialization.[6] The probability of all agents cooperating is very small at initialization, so the expected return of any agent i acting optimally (*e.g.* pushing the boulder) has lower expected return than acting suboptimally (*e.g.* avoiding the boulder) [4, 29]. To ensure the task setting aligns with our problem of interest, we rescale the reward function in some environments so that the expected policy gradient under uniformly initialized policies encourages cooperation. Without this adjustment, these games would not reflect the failure mode we aim to study and would be unsuitable for testing our hypotheses.

[6] Christianos et al. [4] show that on-policy policy gradient algorithms like MAPPO and MAA2C consistently converge suboptimally in the same tasks we consider.

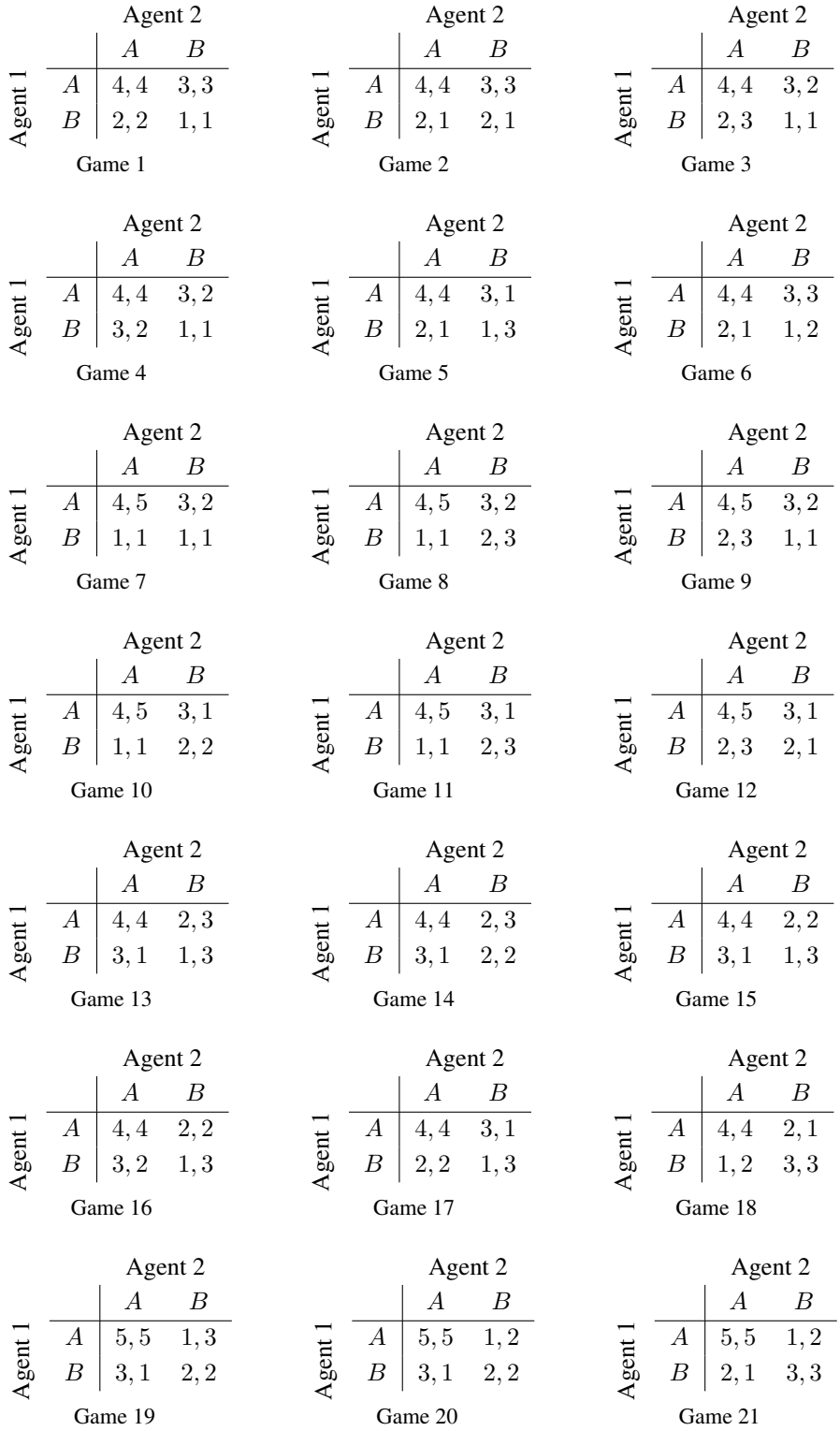


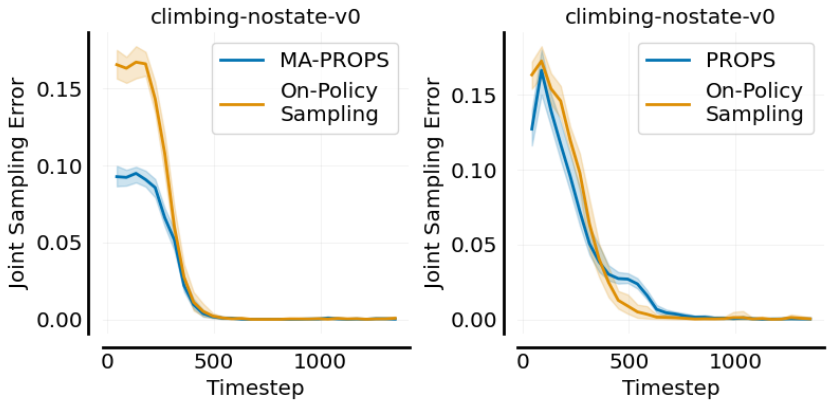
Figure 9: All structurally distinct 2×2 no-conflict matrix games from Section 11.2.1 of Albrecht et al. [52]. Each cell shows the reward pair (r_1, r_2) for Agents 1 and 2. In all games, the optimal outcome is (A, A). To ensure the true policy gradient w.r.t. uniformly random policies increases the probability of the optimal outcome, we change the reward associated with the optimal outcome to $(4, 5)$ in games 7-12 and $(5, 5)$ in games 19-21.

- **LBF:** We reduce the penalty for failed cooperation from -0.1 to -0.015.
- **BoulderPush:** We reduce the penalty for failed cooperation from -0.1 to -0.02.
- 3×3 **Climbing game:** We show the original Climbing game rewards in Fig. 7a and our modified Climbing game in Fig. 7b.
- 3×3 **Penalty game:** We show the original Penalty game rewards in Fig. 8a and our modified Penalty game in Fig. 8b.
- 2×2 **no-conflict matrix games:** In all games, the optimal outcome originally has reward $4, 4$. In games 7-12, we change this reward to $4, 5$. In games 19-21, we change it to $5, 5$.

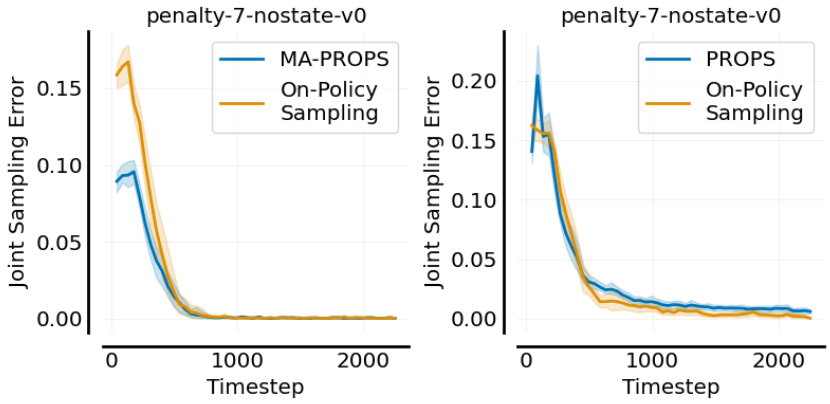
E Additional Experiments

In this section, we provide additional experiments excluded from main paper due to space constraints:

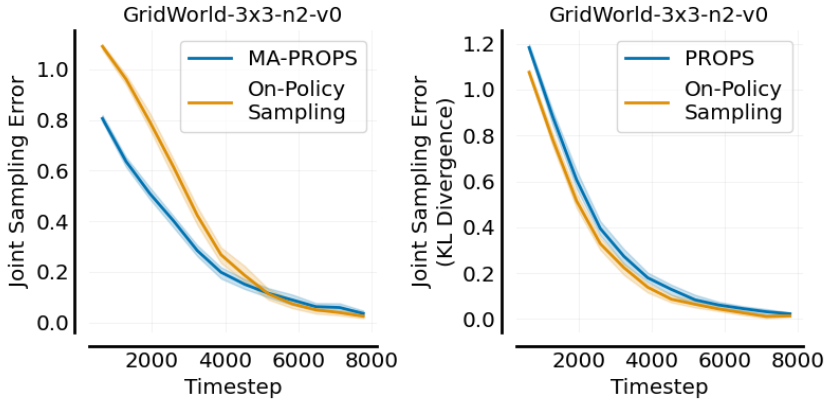
1. Sampling error curves for RL training in GridWorld, Climbing game, and Penalty game (Fig. 10).
2. Training curves for all 21 distinct 2×2 no-conflict matrix games using MAPPO (Fig. 11) and IPPO (Fig. 12)
3. Sampling error curves for RL training in all 21 distinct 2×2 no-conflict matrix games (Fig. 13 and Fig. 14).
4. Training curves for BoulderPush and Level-based foraging tasks using IPPO (Fig. 15).



(a) Climbing Game



(b) Penalty Game



(c) Grid World

Figure 10: Joint sampling error during training in GridWorld, Climbing, and Penalty games. Solid curves denote means over 100 seeds. Shaded regions denote 95% bootstrap confidence intervals.

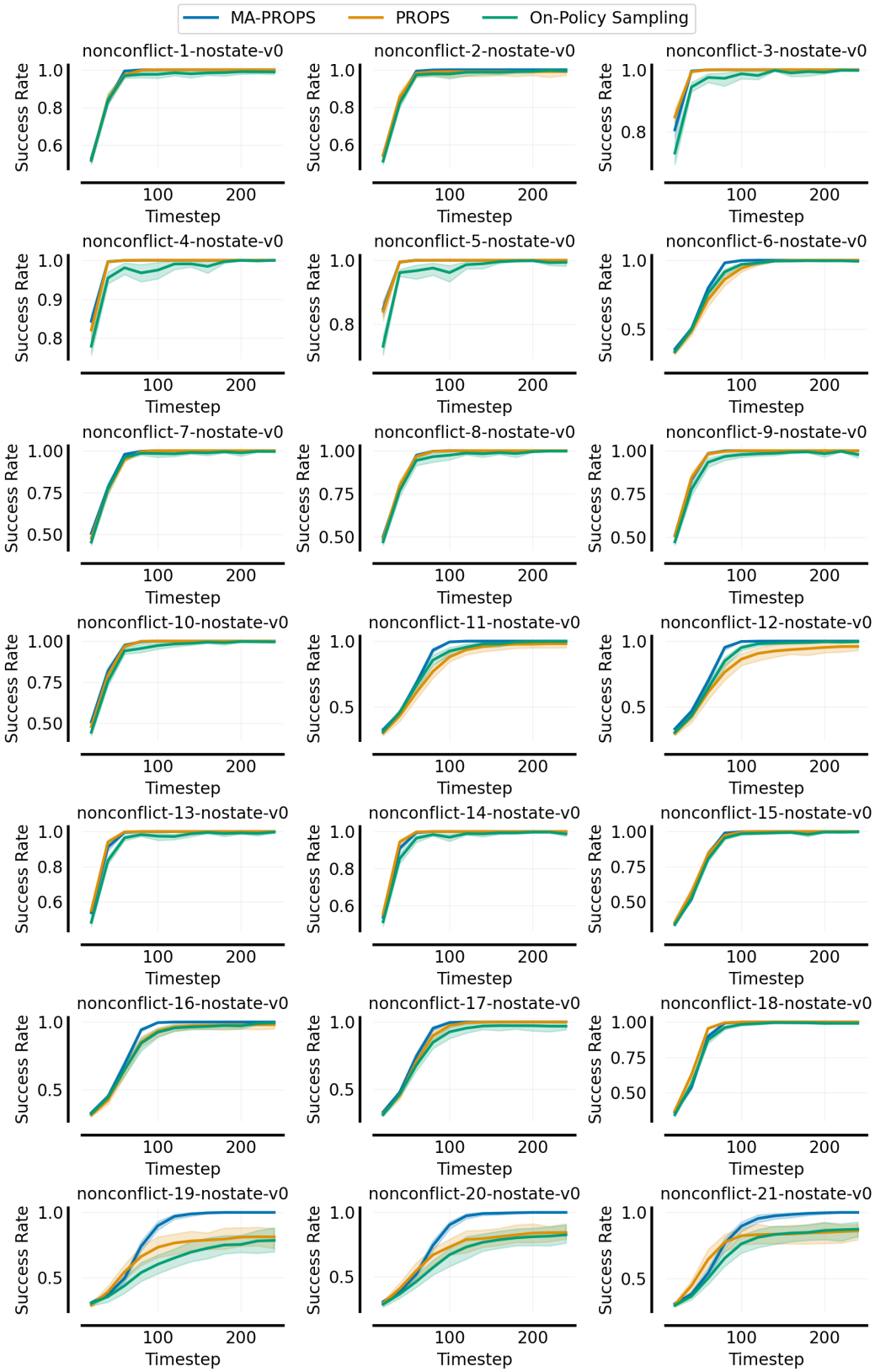


Figure 11: MAPPO success rate during training for all structurally distinct 2×2 no-conflict matrix games. Solid curves denote means over 100 seeds. Shaded regions denote 95% bootstrap confidence intervals.

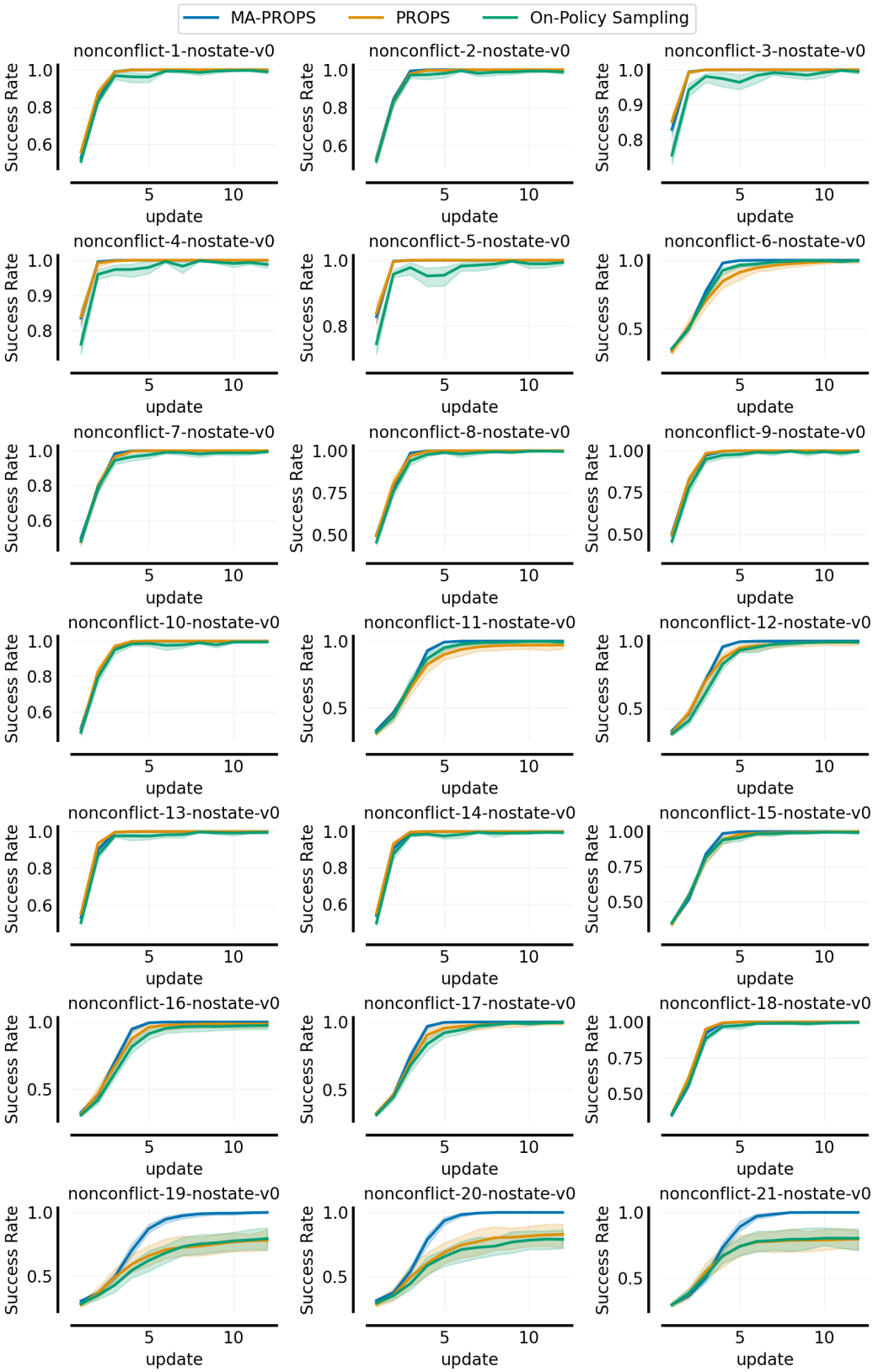


Figure 12: IPPO success rate during training for all structurally distinct 2×2 no-conflict matrix games. Solid curves denote means over 100 seeds. Shaded regions denote 95% bootstrap confidence intervals.

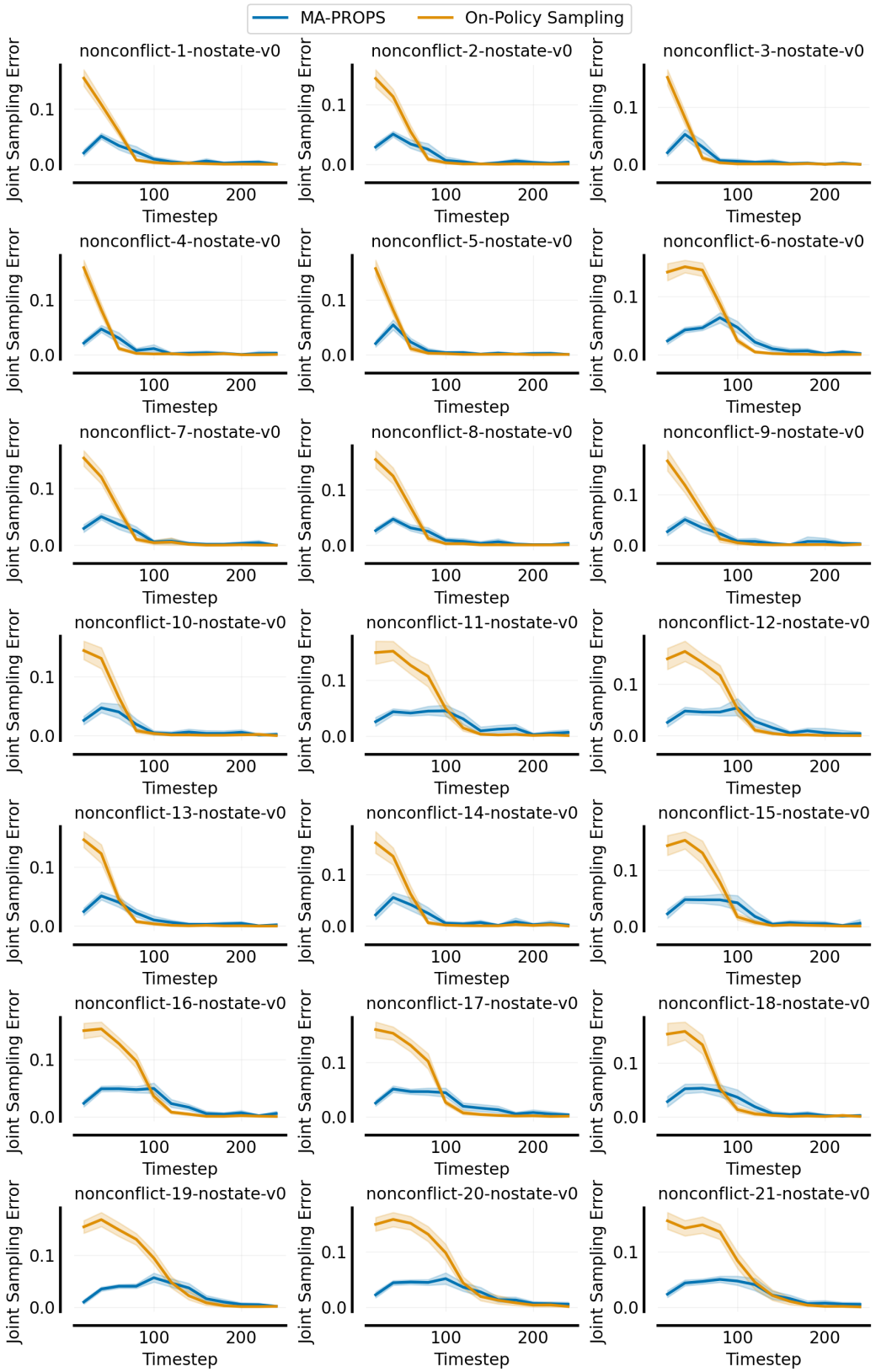


Figure 13: Joint sampling error for MAPPO + PROPS training on all structurally distinct 2×2 no-conflict matrix games. Solid curves denote means over 100 seeds. Shaded regions denote 95% bootstrap confidence intervals. **Takeaway: MA-PROPS reduces sampling error w.r.t. on-policy sampling by a large amount in all games.**

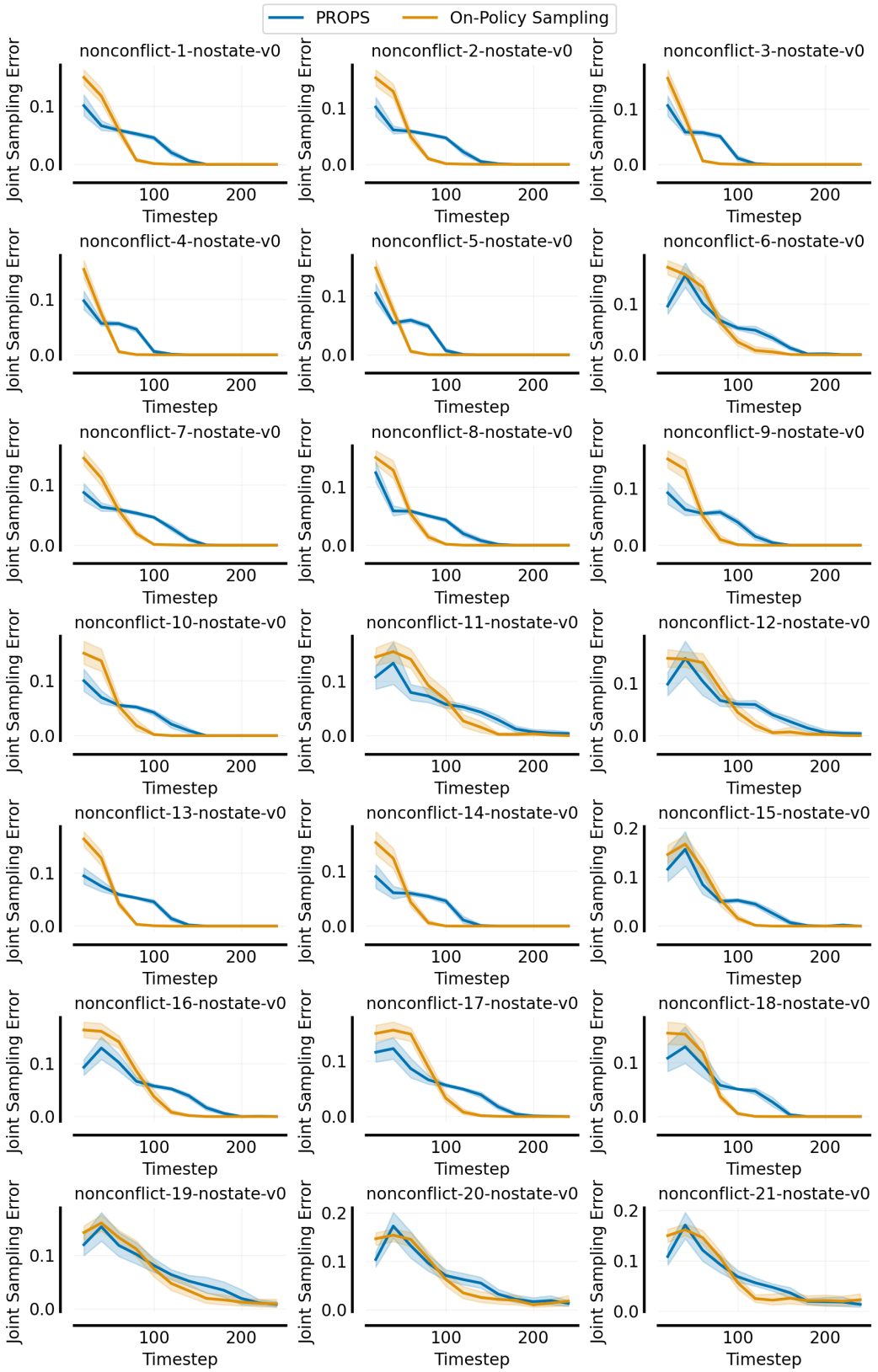


Figure 14: Joint sampling error for MAPPO + PROPS training on all structurally distinct 2×2 no-conflict matrix games. Solid curves denote means over 100 seeds. Shaded regions denote 95% bootstrap confidence intervals. **Takeaway: Early in training, PROPS reduces sampling error w.r.t. on-policy sampling in all games except 19-21. MA-PROPS reduces sampling error by a larger amount in all games (Fig. 13).**

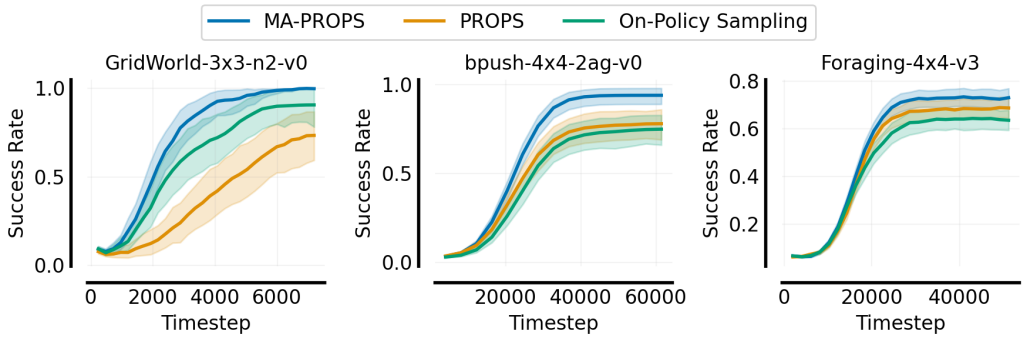


Figure 15: IPPO success rate during training for GridWorld, BoulderPush, and LBF. Solid curves denote means over 100 seeds. Shaded regions denote 95% bootstrap confidence intervals.

Behavior learning rate	$0.3, 0.03, 0.003$
Behavior batch size m	$1, 4$
Behavior KL cutoff	6
Behavior clipping coefficient	$0.3, 1, 10$

Table 1: Hyperparameters used in our hyperparameter sweep for training.

Batch size	See Table. 3
Learning rate	See Table. 3
Number of update epochs	4
Number of minibatch updates per epoch	4
Discount factor γ	0.99
generalized advantage estimation (GAE)	0.95
Advantage normalization	Yes
Loss clip coefficient	0.2
Entropy coefficient	0.01
Value function coefficient	0.5
Gradient clipping (max gradient norm)	0.5
KL cut-off	None
Actor and critic network architectures	Multi-layer perceptron with hidden layers $(64, 64)$
Optimizer	Adam [54]
Number of evaluation episodes	100

Table 2: MAPPO/IPPO hyperparameters across all experiments. We implement MAPPO/IPPO on top of the PPO implementation provided by CleanRL [55].

F Hyperparameters

All hyperparameters used in our experiments are listed in Tables 1, 2, and 3. Our tuning procedure required running approximately one thousand jobs and requires access to many CPUs to do efficiently. We ran all experiments on a computing cluster that enabled us to run several hundred jobs in parallel. The training budget for all experiments are fairly small (less than 100k timesteps), and MA-PROPS and PROPS jobs finish within 1-2 hours. Each job requires 0.6 GB of memory and 1.6 GB of disk.

Game	PPO Batch Size	PPO Learning Rate	MA-PROPS/PROPS Learning Rate	MA-PROPS/PROPS Batch Size
LBF	2048	0.01	0.03	1
BoulderPush	4096	0.003	0.03	1
GridWorld	256	0.01	0.3	1
3×3 matrix games	45	0.1	0.3	1
2×2 matrix games	20	0.1	0.03	1

Table 3: Tuned hyperparameters used in RL training with MA-PROPS and PROPS.