Privacy-Preserving Decentralized Actor-Critic for Cooperative Multi-Agent Reinforcement Learning

Maheed H. Ahmed
Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47906
ahmed237@purdue.edu

Mahsa Ghasemi
Electrical and Computer Engineering
Purdue University
West Lafayette, IN 47906
mahsa@purdue.edu

Abstract

Multi-agent reinforcement learning has a wide range of applications in cooperative settings, but ensuring data privacy among agents is a significant challenge. To address this challenge, we propose Privacy-Preserving Decentralized Actor-Critic (PPDAC), an algorithm that motivates agents to cooperate while maintaining their data privacy. Leveraging trajectory ranking, PPDAC enables the agents to learn a cooperation reward that encourages agents to account for other agents' preferences. Subsequently, each agent trains a policy that maximizes not only its local reward as in independent actor-critic (IAC) but also the cooperation reward, hence, increasing cooperation. Importantly, communication among agents is restricted to their ranking of trajectories that only include public identifiers without any private local data. Moreover, as an additional layer of privacy, the agents can perturb their rankings with the randomized response method. We evaluate PPDAC on the level-based foraging (LBF) environment and a coin-gathering environment. We compare with IAC and Shared Experience Actor-Critic (SEAC) which achieves SOTA results for the LBF environment. The results show that PPDAC consistently outperforms IAC. In addition, PPDAC outperforms SEAC in the coin-gathering environment and achieves similar performance in the LBF environment, all while providing better privacy.

Proceedings of the 27$^{\text{th}}$ International Conference on Artificial Intelligence and Statistics (AISTATS) 2024, Valencia, Spain. PMLR: Volume 238.

1 INTRODUCTION

Deep reinforcement learning algorithms have been successful in solving a wide range of tasks including games (Silver et al., 2017; Mnih et al., 2013) and continuous control tasks (Schulman et al., 2017). In addition, multi-agent reinforcement learning (MARL) has a wide range of applications such as traffic control, multi-robot path planning, and stock market operations (Oroojlooy and Hajinezhad, 2022). Cooperative MARL agents need to be trained simultaneously to achieve a common goal or multiple goals (Oroojlooy and Hajinezhad, 2022); however, both single-agent and multi-agent reinforcement learning can benefit from distributed training as it can drastically reduce the required training time (Mnih et al., 2016; Liang et al., 2018).

Preserving the privacy of data in a distributed learning paradigm is one of the open challenges of MARL alongside other challenges such as scalability, non-stationarity, and partial observability (Gronauer and Diepold, 2022). In this work, we focus on cooperative agents that require maintaining some level of data privacy. We consider the settings where multiple, potentially heterogeneous, agents with private actions and possibly private observations interact simultaneously with an environment and each agent receives a different local reward according to its own task. In this cooperative setting, each agent's goal is to learn a policy that achieves its own objective and supports the other agents' objectives, by maximizing a weighted average of all rewards. Realizing this goal becomes significantly challenging when sharing the actions, observations, or rewards is not allowed due to privacy issues (Oroojlooy and Hajinezhad, 2022).

In this work, we propose the Privacy-Preserving Decentralized Actor-Critic (PPDAC) algorithm which enables a simple yet efficient mechanism for the agents to cooperate while keeping their rewards, actions, and

observations private. It utilizes the actor-critic learning paradigm in which, an actor component learns the policy while a critic component learns the value function (Sutton and Barto, 2018). In PPDAC, each agent learns two critic networks, a critic using its own rewards, and another critic using a cooperation reward designed to encourage actions that benefit other agents. To learn the cooperation reward without sharing private data, we propose trajectory ranking as the mode of communication, relying on the recently shown effectiveness of learning a reward function from trajectory ranking, especially for learning human preferences (Christiano et al., 2017). In particular, each agent estimates the cooperation reward with a parameterized network trained using other agents' aggregated ranking over a shared database of trajectory pairs. A single actor is then trained using an aggregated loss computed using both critic networks. Furthermore, to achieve a higher level of privacy, agents can use the randomized response method (Warner, 1965) to return a random ranking instead of a truthful one with a predetermined probability. We show that this method achieves local differential privacy for the agents' preferences over trajectories. To the best of our knowledge, this is the first algorithm for heterogeneous MARL that proposes trajectory ranking as an incentive for cooperation between agents that wish to keep their data private.

We thoroughly evaluate PPDAC by running simulations in the level-based-foraging environment (Papoudakis et al., 2021) and a newly created coin-gathering environment in six different settings. We compare PPDAC with independent actor-critic (IAC) and Shared Experience Actor-Critic (SEAC) which achieves state-of-the-art results in the LBF environment. The results show a clear advantage for PPDAC over IAC in both environments. In addition, PPDAC outperforms SEAC in the coin-gathering environment and achieves similar performance in the LBF environment without sharing the agents' private data.

The remainder of the paper is organized as follows. Section 2 reviews the existing related work. We provide the necessary technical background in Section 3 and present the proposed PPDAC algorithm in Section 4. We demonstrate and discuss the simulation results in Section 5 and finally, conclude our findings and provide future work directions in Section 6.

2 RELATED WORK

We now overview the research directions, along with the existing work, closely related to this paper.

Centralized Training with Decentralized Execution for Cooperative MARL One common approach to tackling cooperative MARL problems is centralized training with decentralized execution (CTDE) such as value decomposition networks (VDN) (Sunehag et al., 2017), QMIX (Rashid et al., 2020), and multi-agent deep deterministic policy gradient (MADDPG) (Lowe et al., 2017). In this setup, a centralized critic is trained to compute the Q-value function of the joint policy of all agents, and through value factorization, each agent's contribution is used to update its local policy. Each agent executes its local policy independently according to its own observation in a decentralized manner. In recent literature, different value factorization methods are used. For example, Du et al. (2019) proposes learning an intrinsic reward function for each agent with the existence of a centralized critic. Their method merges reward shaping and centralized training in order to learn policies with diversified behavior. Peng et al. (2021) proposed FACtored Multi-Agent Centralized policy gradients (FACMAC) where a centralized factored critic is trained with no constraints on factorization, unlike prior methods. Although CTDE methods achieve state-of-the-art performance on benchmark cooperative multi-agent tasks, a centralized server needs to be trusted with local information including rewards in order to train the centralized critic.

Fully Decentralized Cooperative MARL Decentralized approaches to MARL have more potential for preserving the privacy of each agent's data but require efficient communication between agents (Zhang et al., 2021). Communication can be limited to training time only where agents share observations, rewards, gradients, or any local information to facilitate training. For example, Christianos et al. (2020) proposes Shared Experience Actor-Critic (SEAC) where agents with different local rewards can share their experiences during training which results in better exploration and faster convergence. Although rewards can be different, agents are assumed to have similar observation spaces. Ma et al. (2022) proposes Expectation Alignment as a Multi-Agent Intrinsic Reward (ELIGN) where each agent learns to take actions that match the expectations of its neighbors to maximize a team reward. This is achieved through an intrinsic reward that encourages action alignment. The reward is estimated using the intersection of observations with neighboring agents. In a more restricted setting where rewards are local and private to each agent, Qu et al. (2019) develops the value propagation algorithm where agents communicate with a local neighborhood during training to reach a consensus over the local critic parameters assuming full observ-

ability of the state. In our work, we do not assume full observability, homogeneity of observation and action spaces, or the existence of an intersection in observation. In addition to communication during training, communication between agents both during execution and training has also been studied (Rangwala and Williams, 2020; Zhang et al., 2019; Jiang and Lu, 2018).

Influencial Agents in Cooperative MARL Learning to incentivize other agents has also been studied in the literature. Wang et al. (2022) and Xie et al. (2021) tackle modeling other agents' behavior in order to influence them towards a stable joint policy. Another approach is augmenting the action space of agents to allow each agent to decide gifting rewards to other neighboring agents to incentivize them to do beneficial actions (Yang et al., 2020; Lupu and Precup, 2020).

Privacy-Preserving in RL Sakuma et al. (2008) consider privacy in distributed reinforcement learning. They use cryptographic methods such as encryption and secure function evaluation in order to develop distributed privacy-preserving versions of SARSA and Q-learning in the tabularized form. Vietri et al. (2020) develop the Private Upper Confidence Bound algorithm which is designed to achieve joint differential privacy for a single agent that interacts with multiple users sharing their sensitive data. In a similar setting, Garcelon et al. (2021) achieve local differential privacy through obfuscating data on the user side. In a federated multi-agent reinforcement learning setting, Zhuo et al. (2019) develops deep federated reinforcement learning where agents communicate encrypted Q-values with Gaussian differentials to learn a global Q-network. In a multi-agent planning setting, both Chen et al. (2023) and Ye et al. (2020) develop algorithms that achieve differential privacy. In addition, Karabag et al. (2021) and Liu et al. (2021) study deception in multi-agent planning which is closely related to privacy as agents aim to hide their intentions from other parties observing their interactions with the environment. In this work, we focus on maintaining privacy for agents in a multi-agent reinforcement learning setting where each agent learns a different policy in a decentralized manner.

3 PRELIMINARIES

In this section, we go over the necessary background needed to define the proposed method. We first define the problem as a general-sum Markov game and then discuss existing actor-critic methods implementation.

3.1 Problem Definition

A Markov game $\mathcal{M}$ is defined as the tuple $(\mathcal{N}, \mathcal{S}, \{\mathcal{O}^{(i)}\}_{i\in\mathcal{N}}, \{\mathcal{A}^{(i)}\}_{i\in\mathcal{N}}, \{R^{(i)}\}_{i\in\mathcal{N}}, \mathcal{T})$ where $\mathcal{N} = \{1, ..., N\}$ is the set of N agents, $\mathcal{S}$ is the set of states that the environment could be in, $\mathcal{O}^{(i)} : \mathcal{S} \to \mathbb{R}^d$ maps the set of states to the d-dimensional view of agent i, $\mathcal{A}^{(i)}$ is the set of possible actions for agent i, $R^{(i)} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ is the reward function for agent i, and $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$ is the transition probability where $\mathcal{A} = \mathcal{A}^1 \times ... \times \mathcal{A}^N$ and $\Delta(\mathcal{S})$ is the set of probability distributions over $\mathcal{S}$.

Each agent has its own policy $\pi^{(i)} : \mathcal{O}^{(i)} \to \Delta(\mathcal{A}^{(i)})$ that maps its observation to a probability distribution over actions. The discounted return of agent i for a given trajectory $(s_0, a_0, s_1, a_1, ...)$ is defined as $G^{(i)} = \sum_{t=0}^{\infty} \gamma^t R^{(i)}(s_t, a_t)$, where $\gamma \in (0, 1)$ is the discount factor, and a_t is the joint action of all agents at time t. In a centralized cooperative setting, the goal is to find a joint policy $\pi = (\pi^{(1)}, ..., \pi^{(N)})$ that maximizes the total expected discounted return of all agents. However, due to the exponentially growing observation and action spaces with the increasing number of agents, we consider a decentralized case where the goal of agent i is to learn a policy that maximizes $\mathbb{E}_{\pi^{(i)}}\left[G^{(i)} + \lambda G^{(-i)} \middle| \pi^{(-i)}\right]$ where $\pi^{(-i)}$ is the joint policy of the other agents, i.e., agents$\{j \in \mathcal{N} : j \neq i\}$, $G^{(-i)}$ is the average discounted return of said agents, λ is the cooperation factor, and $\mathbb{E}_\pi[.]$ is the expectation of a random variable if the policy π is followed under transition probabilities defined by $\mathcal{T} : \mathcal{S} \times \mathcal{A} \to \Delta(\mathcal{S})$.

We assume that every agent views other agents' policies as part of the transition dynamics of the environment and that it does not change drastically. This means that for a single agent, the transition probability $P(s_{t+1} = s' | a_t^{(i)} = a^{(i)}, s_t = s) = \sum_{a^{-i}\in\mathcal{A}^{-i}} \pi^{(-i)}(a^{-i}|s)\mathcal{T}(s, (a^{(i)}, a^{-i}), s')$ appears fixed during a small number of steps. This assumes that the environment appears stationary and allows agents to learn by simply forgetting old timesteps where the environment dynamics were different. This approach is similar to the family of forget algorithms discussed by Hernandez-Leal et al. (2017) on how to deal with non-stationarity in MARL.

3.2 Independet Actor-Critic for MARL

We begin by discussing actor-critic methods for the single-agent case. Actor-critic methods are a subset of model-free policy gradient algorithms. Policy gradient algorithms are based on the REINFORCE algorithm (Williams, 1992) which trains a parameterized policy π_ϕ with parameters ϕ to maximize the expected returns of an agent. Formally, the objective function is defined as $\mathcal{L}(\phi) = \mathbb{E}_{\pi_\phi}[G_t \log \pi_\phi(a_t|s_t)]$ where G_t is the

discounted return at time step t. The parameters ϕ are updated by performing gradient ascent in the direction $\nabla_\phi \log \pi_\phi(a_t|s_t)G_t$ using samples collected from interacting with the environment. REINFORCE with a baseline (Williams, 1992) reduces variance in gradient estimation by subtracting a baseline independent of the actions from G_t such that the gradient is $\nabla_\phi \log \pi_\phi(a_t|s_t)(G_t - b(s_t))$.

In actor-critic methods, a critic V_θ^π parameterized by θ acts as a baseline. The critic is trained to estimate the expected returns of the current policy starting from a given state or observation (Sutton and Barto, 2018). The expected returns of a policy is referred to as the value function of a policy and is defined as $V^\pi(s) = \mathbb{E}_\pi[G_t|s_t = s] = \mathbb{E}_\pi[r_t + \mathbb{E}_{s'\sim\mathcal{T}} V^\pi(s')]$. The gradient of loss with respect to the policy for an n-step actor-critic becomes $\nabla_\phi \log \pi_\phi(a_t|s_t)(\sum_{k=0}^{n-1} \gamma^k r_k + \gamma^n V_\theta^\pi(s_{t+n}) - V_\theta^\pi(s_t))$.

The most natural extension of actor-critic algorithms to MARL is independent actor-critic (IAC) (Foerster et al., 2018) where each agent trains an actor and a critic independently using its local observations, actions, and rewards. Assuming partial observability, a critic $V_{\theta_i}^{\pi^{(i)}}$ is trained per agent to estimate the expected local returns from a given observation. The critic parameters are updated using batches of samples collected from the environment through minimizing the loss function defined in Equation (1) where the $\mathbf{V}_{\theta_i}^{\pi^{(i)}}$ is a vector defined as $\mathbf{V}_{\theta_i}^{\pi^{(i)}}(t) = V_{\theta_i}^{\pi^{(i)}}(o_t^{(i)})$ and the targets vector $\mathbf{Y}^{(i)}$ is defined in Equation (2).

$$\mathcal{L}(\theta_i) = \left|\left|\mathbf{V}_{\theta_i}^{\pi^{(i)}} - \mathbf{Y}^{(i)}\right|\right|_2^2 \tag{1}$$

$$\mathbf{Y}^{(i)}(t) = \sum_{k=0}^{n-1} \gamma^k r_{t+k}^{(i)} + \gamma^n V_{\theta_i}^{\pi^{(i)}}\left(o_{t+n}^{(i)}\right) \tag{2}$$

For each agent, a policy $\pi_{\phi_i}^{(i)}$ is trained to map the observation to a probability distribution over actions by minimizing the loss defined in Equation (3) where the n-step estimate of the advantage function $A^{(i)}\left(o_t^{(i)}, a_t^{(i)}\right)$ is defined in Equation (4).

$$\mathcal{L}(\phi_i) = -\log \pi_{\phi_i}^{(i)}\left(a_t^{(i)}\middle|o_t^{(i)}\right) A^{(i)}\left(o_t^{(i)}, a_t^{(i)}\right) \tag{3}$$

$$\begin{aligned} A^{(i)}\left(o_t^{(i)}, a_t^{(i)}\right) = &\sum_{k=0}^{n-1} \gamma^k r_{t+k}^{(i)} + \gamma^n V_{\theta_i}^{\pi^{(i)}}(o_{t+n}^{(i)}) \\ &- V_{\theta_i}^{\pi^{(i)}}(o_t^{(i)}) \end{aligned} \tag{4}$$

We model PPDAC as an extension to the advantage actor-critic (Mnih et al., 2016) which is defined by the loss in Equations (1) and (3). However, there exist several extensions to on-policy actor-critic methods such as Trust Region Policy Optimization (TRPO) (Schulman et al., 2015) and Proximal Policy Optimization (PPO) (Schulman et al., 2017) which can also be used in the updates of the PPDAC algorithm.

4 PRIVACY-PRESERVING DECENTRALIZED ACTOR-CRITIC

In this section, we define the structure of the Privacy-Preserving Decentralized Actor-Critic (PPDAC) algorithm and analyze its privacy guarantees. There are two key components to PPDAC: learning cooperation through estimating a cooperation reward and training the actor, the local critic, and the cooperation critic through minimizing the PPDAC loss function. An overview of the main modules of the PPDAC architecture for the simple case of 1-step return updates is provided in Figure 1. The algorithm's pseudocode for a single agent with n-step returns is shown in Algorithm 1.

4.1 Learning to Cooperate

In order to learn to cooperate, each agent learns a cooperation reward estimator $\hat{r}_{\nu_i}^{(i)} : \mathcal{O}^i \times \mathcal{A}^i \to \mathbb{R}$ based on the preferences of other agents. We use a similar method to the method proposed by Christiano et al. (2017) to learn a reward function from human preference. First, we assume that with each observation the environment outputs a unique timestep t that acts as an identifier of the current sample. This unique identifier can be used by the agents to retrieve the true local data for a given set of timesteps. A centralized trajectory server creates pairs of timestep sequences of a fixed length m. Each timestep sequence $\sigma = t_1, ..., t_m$ corresponds to a trajectory where t_1 is the start time of the trajectory and t_m is the end time of the trajectory. Whenever the trajectory server creates a batch of pairs, it queries all agents for their preferences and saves the agents' preferences along with the trajectory pairs in a shared buffer. Given a trajectory, each agent can retrieve the true samples (o_t^i, a_t^i, r_t^i) corresponding to the trajectory from its replay buffer. Subsequently, for a given trajectory pair σ_1 and σ_2, each agent computes a score for each trajectory using the scoring function defined in Section 4.2. The agent then assigns to each pair 0.5 if the difference between their scores is less than a predefined threshold ρ, 0 if σ_1 is preferred, or 1 if σ_2 is preferred.

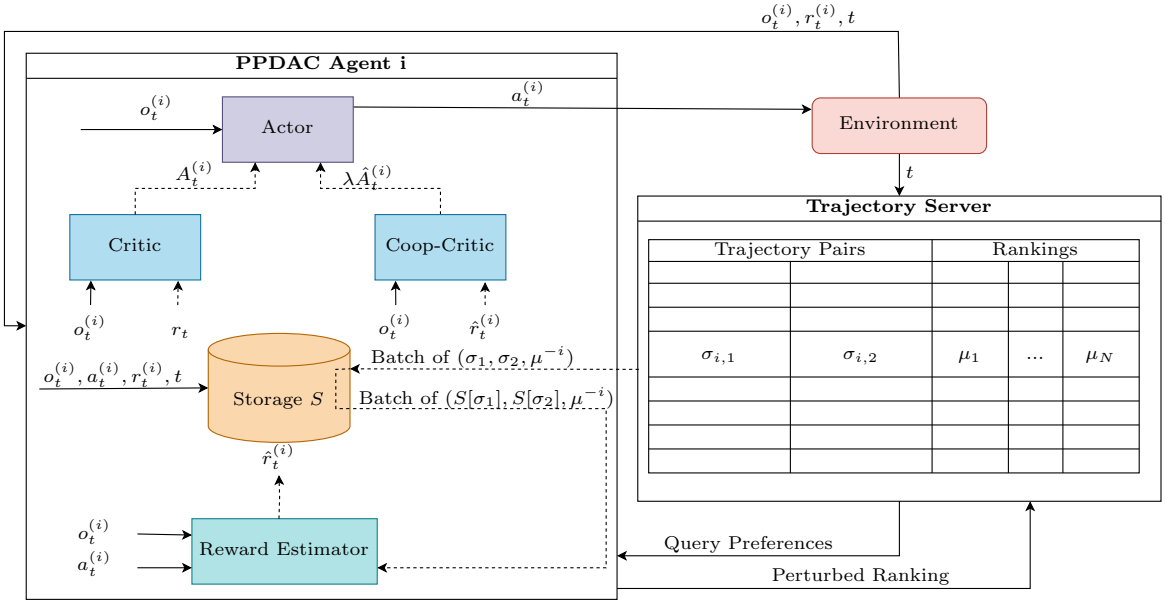


Figure 1: Overview of a PPDAC agent's interactions with the environment and the trajectory server in the simple case of 1-step return updates. The agent chooses an action based on the current observation provided by the environment and receives the next observation, reward, and unique ID t. The agent computes the loss and updates the actor, the critic, and the cooperation critic using the inputs on the dotted line. One of these inputs is the cooperation reward computed by the reward estimator. Data sent by the environment is saved in the local storage S. In parallel, the trajectory server creates trajectory pairs from the unique IDs t, queries the agent for its preference, and receives the perturbed ranking from the agent. The agent then samples a batch of pairs from the server, retrieves the true data from the storage S, and updates the reward estimator. (Note that to reduce clutter some connectors are disconnected but the labels on the arrows are the same)

Using the mean of the preferences of other agents in the trajectory server, each agent i then trains the cooperation reward estimator $\hat{r}_{\nu_i}^{(i)}$ using only their local observation, local action, and the preferences of other agents. The loss function of the reward estimator is based on the Bradley-Terry-Luce (BTL) model (Bradley and Terry, 1952) which models the probability of getting a specific ranking of trajectories based on the cumulative reward achieved during both trajectories. The reward estimator is trained by sampling random batches from the trajectory server using the loss function defined in Equations (5) and (6) where μ^{-i} is the average preference of other agents and ν_i represents the parameters of $\hat{r}_{\nu_i}^{(i)}$.

$$\hat{P}^{(i)}\left[\sigma_1 \succ \sigma_2\right] = \frac{\sum_{t\in\sigma_1} \exp \hat{r}_{\nu_i}^{(i)}(o_t^i, a_t^i)}{\sum_{t\in\sigma_1} \exp \hat{r}_{\nu_i}^{(i)}(o_t^i, a_t^i) + \sum_{t\in\sigma_2} \exp \hat{r}_{\nu_i}^{(i)}(o_t^i, a_t^i)} \tag{5}$$

$$\begin{aligned}\mathcal{L}(\nu_i, \sigma_1, \sigma_2, \mu^{-i}) = \\ -\left(1-\mu^{-i}\right)\log \hat{P}^{(i)}\left[\sigma_1 \succ \sigma_2\right] \\ -\mu^{-i}\log \hat{P}^{(i)}\left[\sigma_2 \succ \sigma_1\right]\end{aligned} \tag{6}$$

In order to maintain the privacy of preferences, each agent uses a randomized response method as suggested by Warner (1965). The agent first samples a number uniformly $u \in [0, 1]$. If $u \leq \zeta$ where ζ is the perturbation noise, the agent returns a random vote, otherwise, the agent ranks the trajectories truthfully. For $\zeta = 0$, the agents are always returning a truthful response with no privacy over its rankings. On the other end of the spectrum for $\zeta = 1$, agents always return a random response maintaining full privacy of their data, but sharing no useful information.

4.2 PPDAC Loss Function

In addition to the cooperation reward estimator, each agent trains two critics by minimizing the loss function defined in Equation (1) where each critic uses different rewards. The local critic $V_{\theta_i}^{\pi^{(i)}}$ is trained similarly to IAC using the local reward samples r_t. The cooperation critic $V_{\omega_i}^{\pi^{(i)}}$ parameterized by ω_i is trained using the cooperation reward samples $\hat{r}_{\nu_i}^{(i)}\left(o_t^i, a_t^i\right)$. The actor of each agent is updated using the loss function defined in Equation (7) where the cooperation factor λ is a hyper-parameter. The actor loss function includes the advantage computed using both critics as

in Equations (4) and (8).

$$\mathcal{L}(\phi_i) = -\log \pi_{\phi_i}^{(i)}\left(a_t^{(i)}\middle|o_t^{(i)}\right) \times \left(A^{(i)}\left(o_t^{(i)}, a_t^{(i)}\right) + \lambda \hat{A}^{(i)}\left(o_t^{(i)}, a_t^{(i)}\right)\right) \tag{7}$$

$$\hat{A}^{(i)}\left(o_t^{(i)}, a_t^{(i)}\right) = \sum_{k=0}^{n-1} \gamma^k \hat{r}_{\nu_i}^{(i)}\left(o_k^{(i)}, a_k^{(i)}\right) + \gamma^n V_{\omega_i}^{\pi^{(i)}}(o_{t+n}^{(i)}) - V_{\omega_i}^{\pi^{(i)}}(o_t^{(i)}) \tag{8}$$

When ranking trajectories, the scoring function defined in Equation (9) is used. The function assigns a score to the trajectory as the sum of the rewards received during the trajectory in addition to the expected discounted returns starting from the last observation which is estimated using the local critic. This inclusion of the expected discounted return of the final observation in the score of a trajectory gives a higher score for trajectories with a final observation that is likely to yield a positive local reward in future timesteps.

$$score^{(i)}(\sigma) = \sum_{t\in\sigma} r_t^{(i)} + V_{\theta_i}^{\pi^{(i)}}(o_{t_m}^{(i)}) \tag{9}$$

4.3 Local Differential Privacy of PPDAC

We study the local differential privacy of PPDAC in the case where agents' observations are disjoint, i.e., an agent does not get a partial observation of other agents' view of the environment. Since every agent can only observe other agents' rankings, it is sufficient to show that PPDAC maintains the privacy of agents' true ranking. Local differential privacy is formally defined as follows (Duchi et al., 2013).

Definition 1. *A randomized algorithm $\mathcal{F}$ satisfies ϵ-local differential privacy if and only if for any two pairs of inputs d_i and d_j and for any output $A \in Range(\mathcal{F})$, the inequality $Pr[\mathcal{F}(d_i) = A] \leq e^{\epsilon} Pr[\mathcal{F}(d_j) = A]$ holds.*

For PPDAC, the randomization step is an extended Warner's randomized response method with three elements (Ma and Wang, 2021). The set of possible inputs and outputs of the randomization step is $\{0, 1, 0.5\}$ which corresponds to an agent's ranking over two trajectories. In PPDAC, the truthful response is returned with probability $Pr[\mathcal{F}(d_i) = d_i] = (1-\zeta)+\frac{1}{3}\zeta = 1-\frac{2}{3}\zeta$, while the other two responses are returned with probability $Pr[\mathcal{F}(d_i) = d_j] = \frac{1}{3}\zeta$ where $j \neq i$. Given this randomization scheme in PPDAC, we present the following Theorem.

Algorithm 1 PPDAC Algorithm

Initialize local critic network $V_{\theta_i}^{\pi^{(i)}}$, cooperation critic network $V_{\omega_i}^{\pi^{(i)}}$, actor network $\pi_{\phi_i}^{(i)}$, and the reward estimator $\hat{r}_{\nu_i}^{(i)}$.
Get initial observation $o_0^{(i)}$ from the environment.
for $t = 0$ to *training_steps* **do**
  Pick action $a_t^{(i)}$ using policy $\pi_{\phi_i}^{(i)}$.
  Execute action $a_t^{(i)}$ and observe next observation $o_{t+1}^{(i)}$ and reward $r_t^{(i)}$.
  for each query from the trajectory server **do**
    Sample a uniform random variable $u \in [0, 1]$.
    if $u \leq \zeta$ **then**
      Respond with a random ranking.
    else
      Respond with 0.5 if $|score^{(i)}(\sigma_1) - score^{(i)}(\sigma_2)| \leq \rho$, 0 if σ_1 preferred, or 1 otherwise.
    end if
  end for
  if $t \mod n == 0$ **then**
    Perform gradient descent to update all networks using the loss functions defined in Equations (6), (1), and (7).
  end if
end for

Theorem 1. *PPDAC satisfies ϵ-local differential privacy for the agents' rankings with $\epsilon = \ln\left(\frac{3-2\zeta}{\zeta}\right)$.*

Proof. For $\zeta \in [0, 1]$, it is clear that $\frac{1}{3}\zeta \leq \frac{1}{3} \leq 1 - \frac{2}{3}\zeta$. Hence, it is sufficient to show that $Pr[\mathcal{F}(d_i) = d_i] \leq e^{\ln\left(\frac{3-2\zeta}{\zeta}\right)} Pr[\mathcal{F}(d_i) = d_j]$ for $j \neq i$ as the other direction of the inequality is trivial given that $Pr[\mathcal{F}(d_i) = d_j] \leq Pr[\mathcal{F}(d_i) = d_i]$ and $e^{\epsilon} \geq 1$, $\forall \epsilon \geq 0$. The probability $Pr[\mathcal{F}(d_i) = d_i]$ can be expressed as

$$Pr[\mathcal{F}(d_i) = d_i] = 1 - \frac{2}{3}\zeta = \frac{3-2\zeta}{\zeta} \times \frac{1}{3}\zeta$$

Therefore, $Pr[\mathcal{F}(d_i) = d_i] = e^{\ln\left(\frac{3-2\zeta}{\zeta}\right)} Pr[\mathcal{F}(d_i) = d_j]$. □

If the disjoint observation assumption is dropped, an agent's rewards can be susceptible to more sophisticated longitudinal attacks that can deduce rewards from multiple rankings of similar observations. In addition, other agents can deduce the agent's intentions by observing the results of the agent's interactions with the environment. To address the former problem, a randomized response that can address longitudinal attacks such as RAPPOR (Erlingsson et al., 2014) could be used. For the latter problem, learning deception (Karabag et al., 2021; Liu et al., 2021) would be required. We leave this for future work.

5 EXPERIMENTS

We evaluate the proposed method in six different settings across two simulation environments, a newly created environment called coin-gathering and the level-based foraging (LBF) environment (Papoudakis et al., 2021). We compare with two baselines, independent actor-critic (IAC) and Shared Experience Actor-Critic (SEAC) (Christianos et al., 2020) which achieves state-of-the-art results in the LBF environment. Additional simulations and analysis of PPDAC are presented in the supplementary material.

5.1 Environments

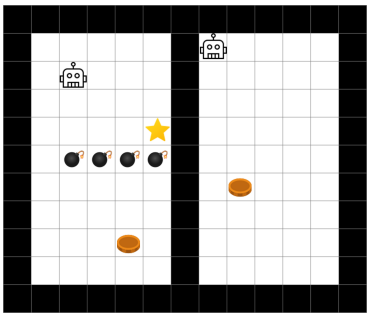
Coin-Gathering (Figure 2a): This environment is developed to mimic systems where each agent is interacting with an isolated sub-system, however, agents' actions affect each other. In this environment, agents exist in a 2D grid where each agent is located in an isolated room that is initially empty. The room gets populated with a coin, a bomb, or a star based on virtual arms that the agents can pull. Picking a coin gives a reward of $+1$, hitting a bomb gives a reward of -0.5, and collecting a star removes all bombs in the room. The agent's observation consists of its (x,y) coordinates, whether a coin exists or not, the relative position of the coin if it exists, whether a star exists, the relative position of the star if it exists, and whether or not each cell of the middle row contains a bomb. Each agent has access to the basic four movement actions in addition to L actions that correspond to pulling a virtual arm. When an agent pulls a virtual arm it has an effect on its room and another effect on other agents' rooms. Each room is divided into a top half and a lower half. Each arm has a predefined probability to either spawn a coin in the half of the room that the agent is not in, spawn a bomb in the middle row of the room blocking the agent's way, spawning a star in the half of the room the agent is in or do nothing. No more coins would be spawned in a room containing a coin until the existing coin gets picked up. In addition, spawning a new star removes the existing star from the room. Each arm has the same effects on the other rooms but with a different probability distribution, i.e., an arm can be beneficial locally to the agent but harmful to other agents.

Level-Based-Foraging (Figure 2b): The LBF environment consists of agents with different levels that get a positive reward when they pick up foods. Each food requires a minimum total level of the agent(s) picking it up. Agents are required to coordinate in order to be able to pick up all foods. The observation of each agent consists of the agents and food positions and levels. Each agent has access to movement actions in the four directions (up, down, left, right) in addition to a do-nothing action and an action that attempts to load food from an adjacent cell. Whenever one or more agents pick up a food item, each agent gets a fraction of the reward proportional to the ratio between the agent's level and the total level of agents that picked up the item. The total reward of a single episode is scaled to be at most one. The reward of an episode reflects the fraction of resources picked up.

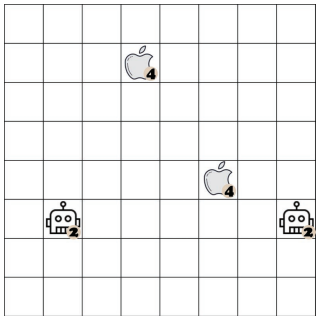
5.2 Implementation Details

Parallel synchronized environments are used to collect the samples needed for training following prior work methods by Mnih et al. (2016) and Christianos et al. (2020). We chose to run 4 environments in parallel. For AC updates, 25 steps are used to estimate the returns. The advantages used to compute the actor loss are normalized before doing the update. This is essential for PPDAC as it scales the advantages of both critics to have similar statistics. However, even for IAC and SEAC, normalization has significantly improved the results for the LBF environment compared to the results obtained by Christianos et al. (2020) which we discuss in the results section. Each of the critics, the actors, and the reward estimators have two hidden layers of size 64 each. Each critic has one linear output representing the estimated value function. Rewards estimators have one *tanh* output and actors have a softmax output layer with k outputs corresponding to k actions. In addition, to reduce variance 3 parallel estimators are trained per agent, and their output is aggregated (Christiano et al., 2017). Finally, entropy regularization is included for the actor loss. The parameters of an agent's actor and critics are updated simultaneously with an Adam optimizer (Kingma and Ba, 2014), while its reward estimators are also updated together. Due to the computational complexity of the experiments, a manual search over hyper-parameters was conducted to find a set of parameters working best for all experiments. IAC, SEAC, and PPDAC use the same values of hyper-parameters for common parameters. The full list of hyper-parameters is included in the Appendix. For SEAC, we use the implementation by Christianos et al. (2020).

In every experiment setting, each algorithm is run three times with different seeds and evaluated by simulating 100 episodes and computing the average total reward. We evaluate an algorithm every 10^4 or 5×10^4 simulation steps depending on the task. The average across the three seeds is plotted along with the 95% confidence region shaded with the same color key. Each run is carried out using 5 cores of an AMD Epyc 7763 "Milan" CPU @ 2.2GHz with allocated memory of 4 GB. We provide an open-source implementation of



(a) Coin-gathering environment: each agent, illustrated with the robot icon, is in a separate room and is required to collect any spawning coins and avoid bombs. An agent can pick a star to remove all bombs in its room.



(b) Level-based-foraging environment (Christianos et al., 2020): agents are illustrated with the robot icon, and the level of each agent and food is included in its bottom right. Agents get a positive reward upon picking any food item.

Figure 2: Environments included in the experiments

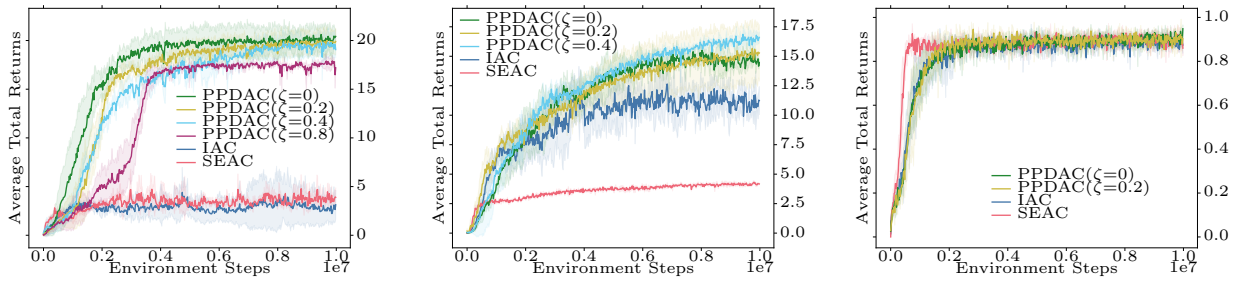
PPDAC in `www.github.com/MaheedHatem/private_coop_marl/`.

5.3 Results

The first experiment shown in Figure 3a studies a simple setting in the coin-gathering environment with two agents where each agent has access to two virtual levers. Each lever has no effect on the agent's room, however, for the other agent's room, one lever always spawns a coin and the other lever always spawns a bomb. Given the limited observation of each agent, IAC and SEAC are unable to find a local policy capable of achieving significant rewards. On the other hand, PPDAC with perturbation up to $\zeta = 0.8$ is capable of converging to a policy that achieves much higher total rewards. As the level of perturbation increases, PPDAC converges to a slightly lower reward with a slight increase in steps required to converge. In the remaining experiments, we only include one or two levels of reasonable perturbation levels. In a more general setting, Figure 3b shows the results of simulations in the coin-gathering environment with three agents and four levers. One of the levers appears to be optimal if viewed locally as it spawns a coin locally with a high probability, however, it spawns a bomb in other rooms with a high probability. The other three levers have a mixed probability of spawning coins, stars, and bombs. The full experiment setting is included in the supplementary material. PPDAC converges to a higher total reward than both IAC and SEAC, with SEAC surprisingly performing worse than IAC. The locally greedy policy adopted by IAC that involves using the first lever appears to be sub-optimal with agents hitting bombs more often. However, PPDAC agents are able to find a better joint policy that is better at creating a bomb-free path and converges to a higher reward.

In the LBF environment, Figure 3c shows the evaluation of PPDAC and the baselines in a setting with two agents and two foods. In this simple setting, all algorithms are able to converge to the same value with SEAC having faster convergence as sharing data increases sample efficiency. Figures 3d and 3e show the results of simulations with three agents and three and four foods, respectively. In this slightly harder setting, our implementations of IAC and PPDAC are able to match the performance of SEAC. This result is a bit different from what was obtained by Christianos et al. (2020) where IAC was unable to match SEAC's performance. We hypothesize that this is due to the advantage normalization step and using a larger number of steps to compute the returns as this is the main difference between our implementation and the implementation of IAC from (Christianos et al., 2020). In addition, SEAC and IAC also achieve higher returns using this modification in the final LBF setting shown in Figure 3f. In this setting, there are two agents with two foods but each food always requires cooperation to be picked up. PPDAC and SEAC achieve a near-optimal value of almost 1 which reflects picking up both foods, outperforming IAC.

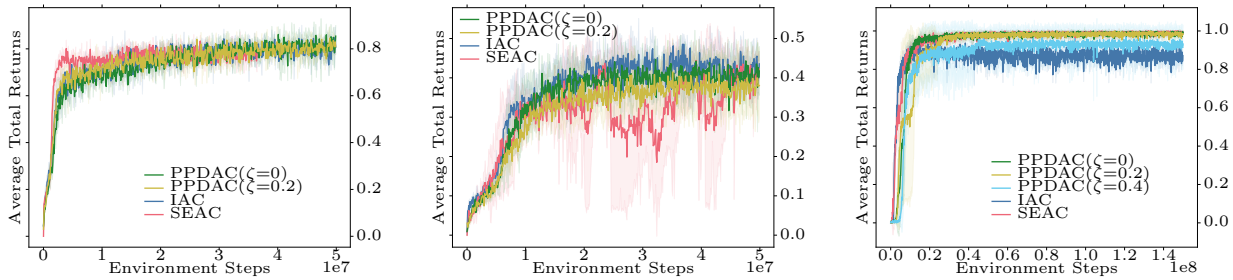
PPDAC has a clear advantage in the coin-gathering environment where the agent's observations are disjoint. In such scenarios, the cooperation reward within PPDAC provides a compelling incentive for agents to collaborate, even when they lack direct observation of the consequences of their actions on others. On the other hand, even when sharing experience, agents do not have a clear way to collaborate. Although SEAC shares experience among agents, the goal of sharing is to enhance exploration and there is no notion of cooperation in the objective function. On the contrary, PPDAC agents take into consideration the rewards of



(a) Coin-gathering: two agents with two locally ineffective levers. Each room is 2×9.

(b) Coin-gathering: three agents with four levers. Each room is 5×12.

(c) LBF: two agents with two foods. Grid size is 12×12.



(d) LBF: three agents with three foods. Grid size is 10×10.

(e) LBF: three agents with four foods. Grid size is 15×15.

(f) LBF: two agents with two foods (Cooperative). Grid size is 8×8.

Figure 3: Average total returns of agents over three random seeds in the coin-gathering and LBF environments. The x-axis shows the number of simulation steps and the y-axis shows the average total rewards received by the agents.

other agents by aligning with their preferences. Furthermore, in the case where a joint observation exists such as in the LBF environment, PPDAC is still able to outperform independent training and match the performance of SEAC where the actions and observations of other agents are shared during training. Notably, with a non-zero value of the perturbation noise ζ, PPDAC agents maintain their performance across both settings. In one case, in the coin-gathering environment with three agents, PPDAC with perturbation ($\zeta = 0.4$) achieves the best performance. This could either stem from the random nature of the training procedure or due to the noise acting as regularization which could improve performance as commonly exercised in deep learning (Noh et al., 2017).

6 CONCLUSION

We proposed Privacy-Preserving Decentralized Actor-Critic (PPDAC), a method for agents in a cooperative multi-agent reinforcement learning setting to collaborate while keeping their data private. By simply ranking a shared database that holds sequences of timestamps, agents can estimate the effect of their actions on other agents. This incentivizes agents to take actions that help increase the overall reward of all agents. In addition to not sharing any of the private data, agents can achieve a higher level of privacy by perturbing their rankings. Through extensive simulations on the coin-gathering and level-based-foraging environments, we show that PPDAC with and without perturbed ranking is able to outperform independent actor-critic (IAC) and shared experience actor-critic (SEAC) agents in settings where agents' observations are disjoint. Furthermore, in settings with overlapping observations, PPDAC learns policies with the same quality as SEAC which performs better than IAC. PPDAC is able to achieve this performance while maintaining the privacy of the agents' data.

Future work could address settings where agents interact asynchronously with an environment. In that setting, a different approach for trajectory ranking would need to be developed. In addition, in a large network where a central database is not feasible, a different communication paradigm would need to be adopted to propagate agents' preferences throughout the network. Finally, when agents observations overlap, developing a randomization technique and studying its privacy guarantees is necessary.

References

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*, 2017.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

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

Afshin Oroojlooy and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *Applied Intelligence*, pages 1–46, 2022.

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.

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*, pages 3053–3062. PMLR, 2018.

Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, pages 1–49, 2022.

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

Paul F Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. Deep reinforcement learning from human preferences. *Advances in neural information processing systems*, 30, 2017.

Stanley L Warner. Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309): 63–69, 1965.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V. Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks (NeurIPS)*, 2021.

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.

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. *The Journal of Machine Learning Research*, 21(1):7234–7284, 2020.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Yali Du, Lei Han, Meng Fang, Ji Liu, Tianhong Dai, and Dacheng Tao. Liir: Learning individual intrinsic reward in multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Bei Peng, Tabish Rashid, Christian Schroeder de Witt, Pierre-Alexandre Kamienny, Philip Torr, Wendelin Böhmer, and Shimon Whiteson. Facmac: Factored multi-agent centralised policy gradients. *Advances in Neural Information Processing Systems*, 34:12208–12221, 2021.

Kaiqing Zhang, Zhuoran Yang, and Tamer Başar. Multi-agent reinforcement learning: A selective overview of theories and algorithms. *Handbook of reinforcement learning and control*, pages 321–384, 2021.

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.

Zixian Ma, Rose Wang, Fei-Fei Li, Michael Bernstein, and Ranjay Krishna. Elign: Expectation alignment as a multi-agent intrinsic reward. *Advances in Neural Information Processing Systems*, 35:8304–8317, 2022.

Chao Qu, Shie Mannor, Huan Xu, Yuan Qi, Le Song, and Junwu Xiong. Value propagation for decentralized networked deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Murtaza Rangwala and Ryan Williams. Learning multi-agent communication through structured attentive reasoning. *Advances in Neural Information Processing Systems*, 33:10088–10098, 2020.

Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient communication in multi-agent reinforcement learning via variance based control. *Advances in Neural Information Processing Systems*, 32, 2019.

Jiechuan Jiang and Zongqing Lu. Learning attentional communication for multi-agent cooperation. *Advances in neural information processing systems*, 31, 2018.

Woodrow Zhouyuan Wang, Andy Shih, Annie Xie, and Dorsa Sadigh. Influencing towards stable multi-agent interactions. In *Conference on robot learning*, pages 1132–1143. PMLR, 2022.

Annie Xie, Dylan Losey, Ryan Tolsma, Chelsea Finn, and Dorsa Sadigh. Learning latent representations to influence multi-agent interaction. In *Conference on robot learning*, pages 575–588. PMLR, 2021.

Jiachen Yang, Ang Li, Mehrdad Farajtabar, Peter Sunehag, Edward Hughes, and Hongyuan Zha. Learning to incentivize other learning agents. *Advances in Neural Information Processing Systems*, 33:15208–15219, 2020.

Andrei Lupu and Doina Precup. Gifting in multi-agent reinforcement learning. In *Proceedings of the 19th International Conference on autonomous agents and multiagent systems*, pages 789–797, 2020.

Jun Sakuma, Shigenobu Kobayashi, and Rebecca N Wright. Privacy-preserving reinforcement learning. In *Proceedings of the 25th international conference on Machine learning*, pages 864–871, 2008.

Giuseppe Vietri, Borja Balle, Akshay Krishnamurthy, and Steven Wu. Private reinforcement learning with pac and regret guarantees. In *International Conference on Machine Learning*, pages 9754–9764. PMLR, 2020.

Evrard Garcelon, Vianney Perchet, Ciara Pike-Burke, and Matteo Pirotta. Local differential privacy for regret minimization in reinforcement learning. *Advances in Neural Information Processing Systems*, 34:10561–10573, 2021.

Hankz Hankui Zhuo, Wenfeng Feng, Yufeng Lin, Qian Xu, and Qiang Yang. Federated deep reinforcement learning. *arXiv preprint arXiv:1901.08277*, 2019.

Bo Chen, Calvin Hawkins, Mustafa O Karabag, Cyrus Neary, Matthew Hale, and Ufuk Topcu. Differential privacy in cooperative multiagent planning. *arXiv preprint arXiv:2301.08811*, 2023.

Dayong Ye, Tianqing Zhu, Sheng Shen, Wanlei Zhou, and S Yu Philip. Differentially private multi-agent planning for logistic-like problems. *IEEE transactions on dependable and secure computing*, 19(2): 1212–1226, 2020.

Mustafa O Karabag, Melkior Ornik, and Ufuk Topcu. Deception in supervisory control. *IEEE Transactions on Automatic Control*, 67(2):738–753, 2021.

Zhengshang Liu, Yue Yang, Tim Miller, and Peta Masters. Deceptive reinforcement learning for privacy-preserving planning. *arXiv preprint arXiv:2102.03022*, 2021.

Pablo Hernandez-Leal, Michael Kaisers, Tim Baarslag, and Enrique Munoz De Cote. A survey of learning in multiagent environments: Dealing with non-stationarity. *arXiv preprint arXiv:1707.09183*, 2017.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8:229–256, 1992.

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.

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.

Ralph Allan Bradley and Milton E Terry. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345, 1952.

John C Duchi, Michael I Jordan, and Martin J Wainwright. Local privacy and statistical minimax rates. In *2013 IEEE 54th Annual Symposium on Foundations of Computer Science*, pages 429–438. IEEE, 2013.

Fei Ma and Ping Wang. Randomized response mechanisms for differential privacy data analysis: Bounds and applications. *arXiv preprint arXiv:2112.07397*, 2021.

Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Hyeonwoo Noh, Tackgeun You, Jonghwan Mun, and Bohyung Han. Regularizing deep neural networks by noise: Its interpretation and optimization. *Advances in neural information processing systems*, 30, 2017.

Checklist

1. For all models and algorithms presented, check if you include:
   (a) A clear description of the mathematical setting, assumptions, algorithm, and/or model. Yes, the description is provided in Section 4.
   (b) An analysis of the properties and complexity (time, space, sample size) of any algorithm. No, an analysis of the complexity is not provided. However, the model's sizes and architecture are clearly described in Section 5.
   (c) (Optional) Anonymized source code, with specification of all dependencies, including external libraries. Yes.

2. For any theoretical claim, check if you include:
   (a) Statements of the full set of assumptions of all theoretical results. Yes in Section 4.3.
   (b) Complete proofs of all theoretical results. Yes.
   (c) Clear explanations of any assumptions. Yes.

3. For all figures and tables that present empirical results, check if you include:
   (a) The code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL). Yes, the code and instructions are included in the supplementary material.
   (b) All the training details (e.g., data splits, hyperparameters, how they were chosen). Yes.
   (c) A clear definition of the specific measure or statistics and error bars (e.g., with respect to the random seed after running experiments multiple times). Yes.
   (d) A description of the computing infrastructure used. (e.g., type of GPUs, internal cluster, or cloud provider). Yes.

4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets, check if you include:
   (a) Citations of the creator If your work uses existing assets. Yes, we use the official implementation of shared-experience actor-critic and cite the authors.
   (b) The license information of the assets, if applicable. Not Applicable.
   (c) New assets either in the supplemental material or as a URL, if applicable. Not Applicable
   (d) Information about consent from data providers/curators. Not Applicable.
   (e) Discussion of sensible content if applicable, e.g., personally identifiable information or offensive content. Not Applicable.

5. If you used crowdsourcing or conducted research with human subjects, check if you include:
   (a) The full text of instructions given to participants and screenshots. Not Applicable.
   (b) Descriptions of potential participant risks, with links to Institutional Review Board (IRB) approvals if applicable. Not Applicable.
   (c) The estimated hourly wage paid to participants and the total amount spent on participant compensation. Not Applicable.

A EXPERIMENTAL SETUP

The full list of hyper-parameters used in the experiments in Section 4 is presented in Table 1. Common hyper-parameters are grouped together and algorithm-specific hyper-parameters are listed after.

Table 1: Hyper-parameters

Common Parameters	
Discount Factor λ	0.99
Number of parallel processes	4
n-steps to estimate returns	25
Value coefficient (multiple to the critic's loss)	0.5
Entropy coefficient	0.01
Learning rate for coin-gathering environment with two agents	1×10^{-4}
Learning rate for other environments	3×10^{-4}
Adam optimizer epsilon	1×10^{-4}
Maximum gradient norm	0.5
Neural networks' hidden layers sizes	$[64, 64]$
SEAC Parameters	
SEAC loss cofficient	1.0
PPDAC Parameters	
Cooperation factor λ	0.5
Number of reward estimators	3
Similarity threshold ρ	0.25
Cooperation value coefficient (multiple to the cooperation critic's loss)	0.25
Reward estimator batch size	32

The full probabilities of the levers in the coin-gathering with three agents (Figure 3b) are given in Table 2. The probabilities of the effect of each lever on the local room are first listed followed by the probabilities of the effect on other rooms.

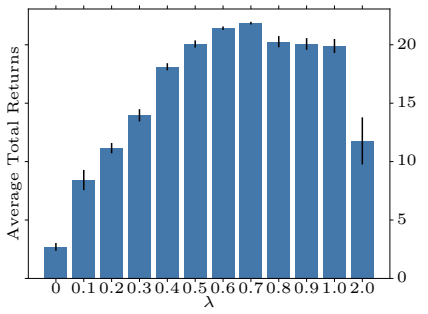
Table 2: Levers effects probabilities in the coin-gathering environment with three agents.

	Spawn a coin	Spawn a bomb	Spawn a star	Do nothing
Lever 1 (locally)	1	0	0	0
Lever 1 (other rooms)	0	0.9	0.0	0.1
Lever 2 (locally)	0.7	0.3	0	0
Lever 2 (other rooms)	0.25	0.05	0.05	0.65
Lever 3 (locally)	0.5	0.25	0	0.25
Lever 3 (other rooms)	0	0.5	0	0.5
Lever 4 (locally)	0	0	0	1
Lever 4 (other rooms)	0	0	0.6	0.4

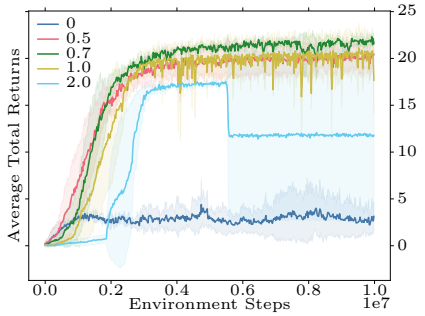
B ADDITIONAL EXPERIMENTS

We run a set of experiments to evaluate the effect of the cooperation factor λ, the trajectory length, and the similarity threshold ρ on the performance of PPDAC. We fix all hyper-parameters to the values shown in Table 1 except for the one under analysis and vary it across a space of a valid range. We analyze the effect of these hyper-parameters in the coin-gathering environment with two agents and two locally-ineffective levers. We evaluate the agents every 1000 steps and average the last 20 evaluations across 3 seeds and report the mean and the 95% confidence region. We also report the full evaluation plot for a set of selected values of each hyper-parameter. The results of this study are shown in Figure 4.

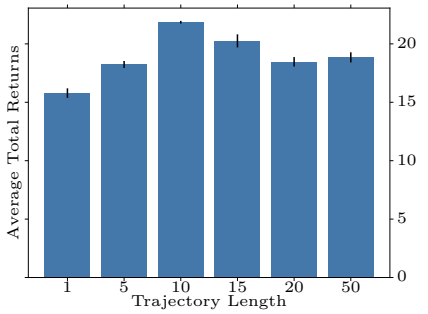
Figures 4a and 4b show the effect of varying the cooperation factor λ starting from 0 where PPDAC is reduced to IAC up to 2. It is clear that for small values of λ, there is a significant increase in returns for small increments.



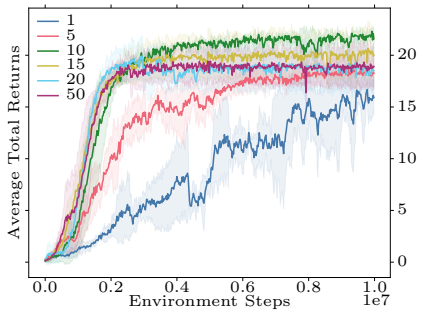
(a) Cooperation factor λ (Aggregated).



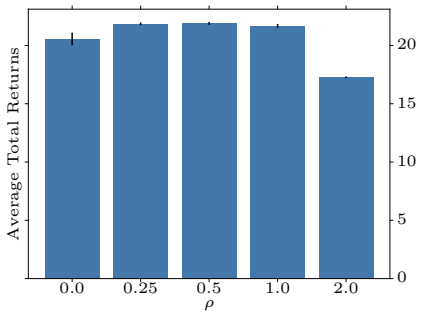
(b) Cooperation factor λ.



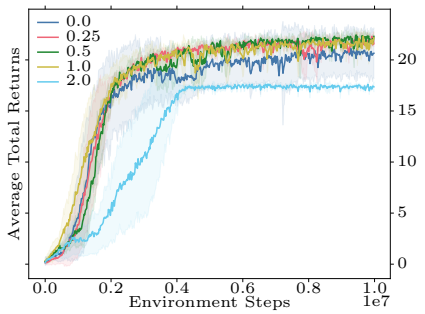
(c) Trajectory length (Aggregated).



(d) Trajectory length.



(e) Similarity threshold ρ (Aggregated).



(f) Similarity threshold ρ.

Figure 4: Analysis of the effect of PPDAC hyper-parameters on the average returns.

However, between 0.5 and 1, there is not a huge variation in the convergence values with the maximum being when $\lambda = 0.7$. In addition, for $\lambda = 2$, where actors would be giving more weight to cooperation, the returns drop significantly. Hence, we suggest tuning λ between 0.5 and 1 when performing hyper-parameter tuning for PPDAC.

Figures 4c and 4d show the effect of the trajectory length used in training the reward estimators. The trajectory length can lie anywhere between 1 and the maximum number of steps per episode, which is 100 in this case. There is a tradeoff incurred when selecting the trajectory length. For the extreme case of length 1, each agent can get feedback for each action from other agents, however, it would require more queries. On the other hand, a larger trajectory length would yield fewer queries but factoring the effect of each action could be more difficult. For a fixed number of queries, PPDAC appears to be a bit sensitive to the value of the trajectory length, hence we suggest assigning computational resources to tuning the trajectory length when the computational budget is limited. For the coin-gathering environment, it seems trajectories of lengths 10 work best. On the contrary, PPDAC seems less sensitive to the similarity threshold ρ as shown in Figures 4e and 4f. We argue that ρ acts as a regularizer since the agents do not pick a preferred trajectory when their scores differ by a small value, hence

other agents would not overfit to the rankings. For $\rho = 0$, PPDAC performs worse than $\rho \in \{0.25, 0.5, 1\}$ with almost no variance between the values in the set. However, for a larger value of $\rho = 2$ the performance drops. The value of ρ depends heavily on the range of the rewards received in the environment. For example, in the LBF environment using $\rho = 1$ does not differentiate between trajectories as the maximum reward received is 1, and all trajectories would get ranked as similar. Hence, we suggest using a small non-zero value for ρ to act as a regularizer.