Assigning Credit with Partial Reward Decoupling in Multi-Agent Proximal Policy Optimization

Aditya Kapoor
Research & Innovation,
Tata Consultancy Services,
Mumbai

Benjamin Freed
Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA

Howie Choset
Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA

Jeff Schneider
Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA

Abstract

Multi-agent proximal policy optimization (MAPPO) has recently demonstrated state-of-the-art performance on challenging multi-agent reinforcement learning tasks. However, MAPPO still struggles with the credit assignment problem, wherein the sheer difficulty in ascribing credit to individual agents' actions scales poorly with team size. In this paper, we propose a multi-agent reinforcement learning algorithm that adapts recent developments in credit assignment to improve upon MAPPO. Our approach leverages partial reward decoupling (PRD), which uses a learned attention mechanism to estimate which of a particular agent's teammates are relevant to its learning updates. We use this estimate to dynamically decompose large groups of agents into smaller, more manageable subgroups. We empirically demonstrate that our approach, PRD-MAPPO, decouples agents from teammates that do not influence their expected future reward, thereby streamlining credit assignment. We additionally show that PRD-MAPPO yields significantly higher data efficiency and asymptotic performance compared to both MAPPO and other state-of-the-art methods across several multi-agent tasks, including StarCraft II. Finally, we propose a version of PRD-MAPPO that is applicable to *shared* reward settings, where PRD was previously not applicable, and empirically show that this also leads to performance improvements over MAPPO.

1 Introduction

Multi-agent reinforcement learning (MARL) has achieved super-human performance on many complex sequential decision-making problems, such as DOTA 2 (Berner et al., 2019), StarCraft II (Vinyals et al., 2019), and capture the flag (Jaderberg et al., 2019). These impressive results, however, come at an immense cost: often, they require millions, if not billions, of time-consuming environmental interactions, and therefore can only be run on high-cost compute clusters.

The *credit assignment problem* contributes to the computational difficulties that plague large-scale MARL algorithms; as the number of agents involved in learning increases, so too does the difficulty of assessing any individual agent's contribution to overall group success (Minsky, 1961; Sutton et al., 1998). While credit assignment already challenges reinforcement learning (RL), it is particularly prominent in large-scale *cooperative* MARL, because, unlike problems in which each agent can act greedily to optimize its own reward, all agents must maximize the total reward earned by the entire group. Therefore, agents must not only consider how their actions influence their own rewards, but also the rewards of every other agent in the group.

A popular class of approaches to MARL are policy-gradient methods, which also suffer from the credit assignment problem. Recent work in improving policy-gradient methods took the approach of developing concepts which were then used to extend the original actor-critic algorithm. These extensions include counterfactual multi-agent policy gradients (COMA) (Foerster et al., 2018), multi-agent game abstraction via graph attention neural networks (G2ANet) (Liu et al., 2020), and partial reward decoupling (PRD) (Freed et al., 2022). **The primary contributions of this paper are 1) the machinery necessary for applying PRD to a state-of-the-art multi-agent policy-gradient method (multi-agent PPO (MAPPO)), and 2) a version of PRD that does not require the environment to provide individual rewards streams for each agent, and instead utilizes a *shared* reward signal.**

PRD simplifies credit assignment by decomposing large cooperative multi-agent problems into smaller decoupled subproblems involving subsets of agents. PRD was applied to the actor-critic algorithm (Freed et al., 2022; Konda & Tsitsiklis, 2000). Meanwhile, significant progress has been made towards improving the data efficiency of policy-gradient algorithms. Most notably, trust-region policy optimization (TRPO) and proximal policy optimization (PPO) improve the data efficiency of actor-critic algorithms by enabling a given batch of on-policy data to be re-used for multiple gradient updates. PPO, in particular, has demonstrated strong performance in multi-agent settings (Yu et al., 2021). However, we argue that because PPO relies on stochastic advantage estimates, it still suffers from the credit assignment problem, and can therefore be improved by incorporating advanced credit assignment strategies.

In this paper, we demonstrate that PRD can be leveraged within the learning updates of PPO for each individual agent, to eliminate the contributions from other irrelevant agents. We find that the resulting algorithm, PRD multi-agent PPO (PRD-MAPPO), exceeds the performance of prior state-of-the-art MARL algorithms such as QMix (Rashid et al., 2018), MAPPO (Yu et al., 2021), LICA (Zhou et al., 2020a), G2ANet (Liu et al., 2020), HAPPO (Kuba et al., 2021) and COMA (Foerster et al., 2018) on a range of multi-agent benchmarks, including StarCraft II. Beyond integrating PRD with MAPPO, we make three key modifications to the original PRD approach proposed by Freed et al. (2022). First, we introduce a "soft" variant that softly re-weights advantage terms in agents' learning updates based on attention weights, rather than the strict decoupling used by Freed et al. (2022). Second, we modify the advantage estimation strategy that allows learning updates to be computed in time that is linear, rather than quadratic, in the number of agents. Finally, we propose a version of PRD-MAPPO that is capable of using *shared* rewards, as opposed to individual agent rewards, thus broadening the range of problems to which our algorithm can be applied.

To gain deeper insight to the source of PRD-MAPPO's improved performance, we visualize the relevant sets identified by PRD, and verify that PRD decomposes multi-agent teams into subsets of agents that should cooperate with one another. Finally, we compare the gradient estimator variance of PRD-MAPPO and MAPPO, and find that PRD-MAPPO indeed tends to avoid the spikes in gradient variance present in MAPPO, helping explain its superior data efficiency and stability.

2 Background

Here we describe our problem formulation as a Markov game. Subsequently, we investigate mathematically how imperfect credit assignment manifests itself in high policy-gradient variance in policy-gradient RL algorithms. Finally, we review PPO and PRD.

2.1 Markov Games

We consider multi-agent sequential decision-making problems that can be modeled as a Markov game. A Markov game is specified by $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \rho_0, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the joint action space, consisting of every possible combination of individual agents' actions, $\mathcal{P}(s_{t+1}|s_t, a_t)$ specifies the state transition probability distribution, $\mathcal{R}(r_t|s_t, a_t)$ specifies the reward distribution, $\rho_0(s_0)$ denotes the initial state distribution, and $\gamma \in (0, 1]$ denotes a discount factor (Littman,

1994). At each timestep $t \in \{0, ..., T\}$, each agent $i \in \{1, ..., M\}$ selects an action independently according to its state-conditioned policy $\pi_i(a_t^{(i)}|s_t^{(i)}; \theta_i)$. Here, T specifies the episode length, M denotes the number of agents, $s_t^{(i)}$ denotes the state information available to agent i, and θ_i denotes the parameters for agent i. Subsequently, individual agent rewards are sampled according to $r_t^{(1)}, ..., r_t^{(M)} \sim \mathcal{R}(\cdot|s_t, a_t)$, and the state transitions according to $s_{t+1} \sim \mathcal{P}(\cdot|s_t, a_t)$.

Although agents receive individual rewards, we are primarily interested in learning *cooperative* behaviors that maximize *total* group return, that is, the sum of all agents' individual rewards across all timesteps. More precisely, we wish to find the optimal agent policy parameters $\theta^* = \{\theta_1^*, ..., \theta_M^*\} = \underset{\theta}{\text{argmax}} J(\theta)$, where

$$J(\theta) = \mathbb{E}\Big[\sum_{t=0}^{T}\sum_{j=1}^{M}\gamma^t r_t^{(j)}\Big|\pi_\theta\Big].$$

This problem formulation is distinct from the "greedy" case, where each agent maximizes its own individual return. In this problem formulation, agents should learn to be altruistic in certain situations, by selecting actions that help maximizes group reward, possibly at the expense of some individual reward.

2.2 Credit Assignment and Policy Gradient Variance

To understand the effects of scaling PPO to large numbers of agents, and how we expect PRD will improve this scaling, we explore how imperfect credit assignment causes difficulties in learning. In this paper, we argue that in policy-gradient algorithms (which includes many popular algorithms such as PPO (Schulman et al., 2017), TRPO (Schulman et al., 2015a), D4PG (Barth-Maron et al., 2018), MADDPG (Lowe et al., 2017), and A3C (Mnih et al., 2016)), the credit assignment problem manifests itself in the form of high variance of advantage estimates. High variance in advantage estimates in turn causes policy gradient estimates to be more noisy, resulting in slower learning.

We consider an actor-critic-style gradient estimate for a single-agent system in its most stripped-down possible form, computed using a single state-action sample:

$$\hat{\nabla}_\theta J(\theta, s, a) = \nabla_\theta \log \pi(a|s)\hat{A}(s, a),$$

where state s is sampled from the state-visitation distribution induced by policy π, action a is sampled from π conditioned on s, and $\hat{A}(s, a)$ is a *stochastic advantage estimate*, which estimates the true *advantage* of taking action a_t in state s_t, and following policy π. The advantage function is typically defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$, where $Q^\pi(s, a)$ and $V^\pi(s)$ are the state-action value function and state-value function, respectively (Sutton et al., 1998). Intuitively, the advantage function measures how much better it is to select a particular action a than a random action from the policy, while in state s. There are many ways to compute $\hat{A}$, generally all involving some error, as the true value functions are unknown (Sutton et al., 1998; Schulman et al., 2015b). If perfect advantage estimation were possible, then so too would be perfect credit assignment, as the advantage function directly measures how a particular action a impacted the total reward obtained by the group.

To gain an understanding of how the gradient variance is impacted by advantage estimator variance, we note that the conditional variance of $\hat{\nabla}_\theta J$, given s and a, is proportional to the variance of $\hat{A}$:

$$\text{Var}(\hat{\nabla}_\theta J|s, a) = (\nabla_\theta \log \pi(a|s)) (\nabla_\theta \log \pi(a|s))^T \text{Var}(\hat{A}|s, a).$$

Moving to a cooperative multi-agent setting, $\hat{A}(s, a)$ is replaced by a summation over individual agents' advantages in the gradient estimate for a particular agent i:

$$\hat{\nabla}_{\theta_i} J(\theta, s, a) = \nabla_{\theta_i} \log \pi_i(a_i|s) \sum_{j=1}^{M} \hat{A}_{ij}(s, a),$$

where $\hat{A}_{ij}(s, a)$ now corresponds to our estimate of how agent i's action influenced the expected future reward of agent j. The summation results from the fact that in the cooperative setting, agent i is no longer interested only in maximizing its own total reward, but is instead interested in maximizing *total group reward*, as discussed in Sec. 2.1. The variance of $\hat{\nabla}_{\theta_i} J$ given s and a now depends on the variance of each individual agent's advantage estimates, as well as the covariance between every pair of agents' advantages. Using Bienaymé's identity, and omitting the arguments to π_i for brevity, we can express this variance as

$$\mathrm{Var}(\hat{\nabla}_{\theta_i} J|s, a) = (\nabla_{\theta_i} \log \pi_i) (\nabla_{\theta_i} \log \pi_i)^T \left(\sum_{j=1}^{M} \mathrm{Var}(\hat{A}_{ij}|s, a) + 2 \sum_{k<j} \mathrm{Cov}(\hat{A}_{ij}, \hat{A}_{ik}|s, a) \right).$$

To simplify analysis, we consider an upper bound on gradient estimator variance, obtained using the Cauchy–Schwarz inequality,

$$\mathrm{Var}(\hat{\nabla}_{\theta_i} J|s, a) \leq (\nabla_{\theta_i} \log \pi_i) (\nabla_{\theta_i} \log \pi_i)^T \left(\sum_{j=1}^{M} \mathrm{Var}(\hat{A}_{ij}|s, a) + 2 \sum_{k<j} \sqrt{\mathrm{Var}(\hat{A}_{ij}|s, a) \mathrm{Var}(\hat{A}_{ik}|s, a)} \right), \tag{1}$$

which can be seen to scale roughly linearly with number of agents, assuming $\mathrm{Var}(\hat{A}_{ij}|s, a)$ is roughly similar for all j. Therefore, to achieve a particular signal-to-noise ratio, more such gradient estimates will need to be averaged together as team size increases, thus increasing the data requirements of the algorithm. This analysis helps explain the mechanism by which improved credit assignment can yield data-efficiency improvements for policy-gradient algorithms, such as A3C (Mnih et al., 2016), TRPO (Schulman et al., 2015a) and PPO (Schulman et al., 2017) algorithms. In particular, our approach aims to eliminate extraneous advantage terms that do not on average contribute to the policy gradient, thereby reducing the number of terms in the summations in (1) and decreasing the total variance. We discuss this further in Sec. 2.4 and 3.

2.3 Proximal Policy Optimization

Earlier policy gradient algorithms, such as actor-critic (AC), suffered from poor data efficiency in part because they were purely on-policy, and therefore required a fresh batch of environmental data to be collected each time a single gradient update was applied to the policy (Konda & Tsitsiklis, 2000; Schulman et al., 2015a; 2017). PPO provides higher data efficiency than AC by enabling multiple policy updates to be performed given a single batch of on-policy data, resulting in larger policy improvements for a fixed amount of data. Given a batch of data, PPO optimizes the policy with respect to a "surrogate" objective that penalizes excessively large changes from the old policy, permitting the agent to perform multiple gradient updates without becoming overly off-policy. Specifically, during each policy optimization step, PPO optimizes the following objective with respect to policy parameters θ,

$$L_{\mathrm{PPO}}(\theta) = \hat{\mathbb{E}} \left[\min \left(\left(r_t(\theta)\hat{A}_t\right), \left(\mathrm{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\right) \right) \right],$$

where $r(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}}(a_t|s_t)}$ is the probability ratio, π_{old} is the data collection policy, π is the updated policy, $\hat{A}_t$ is the stochastic advantage estimate for time t, and $\hat{\mathbb{E}}[\cdot]$ denotes an empirical average over a finite batch of samples (Schulman et al., 2017).

PPO has been recently shown to offer strong performance on multi-agent problems (Yu et al., 2021). However, PPO does not explicitly control the variance of its policy gradient updates, which as we discuss in Sec. 2.2, tends to grow with multi-agent team size. This increased gradient estimate variance means that larger batches of data become necessary to reach a satisfactory signal-to-noise ratio in the learning updates; indeed, (Yu et al., 2021) found that much larger batch sizes were necessary for PPO to perform well on multi-agent tasks. In this work, we seek to combine the data efficiency benefits of PPO with the variance reduction benefits of PRD, to enable further improvements in data efficiency and stability.

2.4 Partial Reward Decoupling

PRD is an approach that enables large multi-agent problems to be dynamically decomposed into smaller subgroups such that cooperation among subgroups yields a fully cooperative group-level solution. In practice, PRD was shown to improve the performance of an AC-style approach, compared to a vanilla AC algorithm. The proposed PRD-AC algorithm was also shown to outperform COMA, a popular method for improved multi-agent credit assignment.

PRD makes use of the fact that, considering two agents i and j at a particular timestep t, if the action of agent i does not influence the expected future reward of agent j, then agent i need not take agent j's rewards into account when computing its advantage estimate for time t, thus streamlining credit assignment. The set of agents whose expected future rewards are impacted by the action of agent i at time t is referred to as the *relevant set* of agent i at time t, denoted $R_i^\pi(s_t, a_t)$. In Freed et al. (2022), a learned value function with an attention mechanism was used to estimate the relevant set for each agent.

There were significant drawbacks to the approach presented by Freed et al. (2022), which we address in this paper. First, PRD was used in the context of the AC algorithm, which has been surpassed by algorithms such as TRPO and PPO. Second, for a problem involving M agents, PRD required M evaluations of the critic function to compute a single agent's gradient update; thus the computational burden for a learning update scaled quadratically with the number of agents. Finally, PRD assumed that the environment provided per-agent reward streams (*i.e.*, provided a scalar reward value to each agent at each timestep). However, many multi-agent problems provide only a single scalar reward for the entire group at each timestep.

3 Improving Proximal Policy Optimization with Partial Reward Decoupling

In this paper, we tackle the credit assignment problem by developing PRD-MAPPO, which leverages a PRD-style decomposition within a PPO learning update to improve credit assignment. More specifically, PRD modifies the original PPO objective by eliminating advantage terms belonging to "irrelevant" agents. As shown by Freed et al. (2022), these irrelevant advantage terms contribute only noise to learning updates, making learning less efficient. PRD uses an attention-based value function to identify when a particular agent's action did not influence another agent's future return, allowing those agents to be decoupled.

To leverage the improved credit assignment capabilities of PRD in PPO, we make two modifications to the standard PPO algorithm: first, we incorporate a learned critic with an attention mechanism. Similar to Freed et al. (2022), the attention weights computed by the critic will be used to estimate the relevant set of agents, as described in Sec. 3.1. Unlike Freed et al. (2022), we modify the critic architecture to allow the relevant sets for each agent to be computed in linear, rather than quadratic time. Second, we modify the surrogate objective of PPO to use the streamlined advantage estimation strategy of PRD, which we describe in Sec. 3.2, using the relevant set estimated using the critic. In this work, we test a novel "soft" relevant set estimation strategy that softly decouples agents, which we find significantly improves performance over a manual thresholding approach as was used by Freed et al. (2022).

3.1 Learned Critics for Relevant Set and Advantage Estimation

Similar to Freed et al. (2022), we use a learned critic function to perform relevant set estimation, albeit with significant modifications. In our approach, each agent i maintains a graph neural network Q function $Q_i^\phi(s_t, a_t)$, which is trained to estimate its expected future individual returns given the current state and actions of all agents. A diagram of our Q function is depicted in Fig. 1. In practice, all agents share the same Q function parameters. Q_i^ϕ takes as input the state information and actions of all agents to estimate a scalar Q value for each agent i.

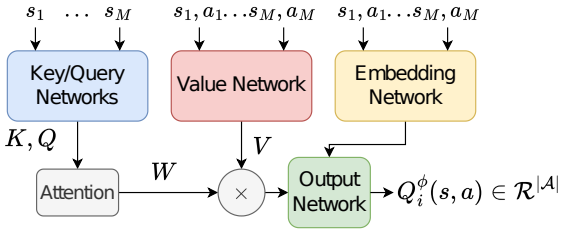

Figure 1: Q and Value Function Network Architecture. Each agent uses states from all agents to compute attention weights for every agent other than itself. These attention weights are then used to aggregate attention values from all agents other than itself. Finally, aggregated attention values for agent i are concatenated either with the embedded state-action vector for agent i (if the network is functioning as a Q function) or the embedded state vector for agent i, (if the network is functioning as a value function). Finally, this is passed through the output network to generate either $Q_i^\phi(s, a)$ or $V_i^\psi(s, a^{\neq i})$.

The Q function contains an attention mechanism that allows it to "shut off" dependence on particular agents' actions. More concretely, the Q network for each agent i uses the states of all agents (including itself) to compute attention weights for all other agents (agents assign an attention weight of 1 to themselves, *i.e.*, $w_{ii}(s_t) = 1$). These attention weights are then used as coefficients to compute a linear combination of attention values computed from agents' states and actions. If a particular attention weight w_{ij} is 0, then any information about agent j's action will not be propagated further through the network, meaning that agent j's action will not influence the final Q estimate for agent i. Once the aggregated value is computed, it is concatenated with an embedding computed from agent i's state and action and passed through a recurrent output network (Fig. 1).

If the learned Q function of agent i at a particular timestep t computes an attention weight of exactly zero for another agent j (*i.e.*, $w_{ij}(s_t) = 0$), then Q_i^ϕ does not depend on $a_t^{(j)}$ given the state of all agents, and we can infer that agent i is outside the relevant set of agent j. As shown by Freed et al. (2022), agents outside the relevant set of agent j do not, on average, contribute to its policy gradient, and may therefore be removed from the policy gradient estimates without introducing bias. In practice, when inferring the relevant sets for each agent, we infer that $i \notin R_j(s_t)$ if $w_{ij}(s_t) < \epsilon$, where $\epsilon > 0$ is a small manually chosen constant. Using this soft attention mechanism, agents cannot assign precisely zero attention weight to any other agent, and therefore cannot guarantee complete independence of the Q function to any particular agent's action. However, we found that in practice, very small attention weights were assigned to irrelevant agents, making this a practical method for relevant set estimation. We explore variants of this decoupling procedure, including a "soft" variant that softly re-weights agents' contributions to learning updates.

Our approach to computing advantage terms for learning updates reduces the computational complexity over (Freed et al., 2022) from quadratic to linear in the number of agents M. To compute the advantage terms required to update the policy of a particular agent i, the original algorithm described by Freed et al. (2022) requires each agent i to estimate the expected future return of each agent j, conditioned on the actions of all agents other than i, for each $j \in R_i(s_t)$. This computation requires (at worst) M calls to the critic for each of the M agents, resulting M^2 total calls during each learning update. Our approach, on the other hand, circumvents with quadratic scaling by maintaining two separate critics; the first is the Q function used for relevant set estimation, described above. The second critic is used solely to provide baseline estimates for advantage function estimation (Schulman et al., 2015b; Konda & Tsitsiklis, 2000). It estimates the *sum* of expected future returns for all agents within agent i's relevant set, conditioned on the state of all agents,

and the actions of all agents other than i. We refer to this critic as the *value* function, rather than the Q function, because it does not depend on the actions of agent i. The value function uses an architecture almost identical to the Q function (Fig. 1), with the one difference that the attention values are concatenated with the embedded *state* of agent i, rather than state-action. Using this value function, computing advantages for all agents requires only M calls (one per agent).

3.2 PRD-MAPPO Parameter Update Rule

We modify the original MAPPO (Yu et al., 2021) objective for each agent i by eliminating the rewards from agents that are outside its relevant set from its advantage estimates. The original MAPPO algorithm optimizes the following objective during each policy parameter update for agent i:

$$L^{(i)}_{\text{MAPPO}} = \hat{\mathbb{E}}\left[\min\left(\left(r^{(i)}_t(\theta_i)\hat{A}_t\right), \left(\text{clip}(r^{(i)}_t(\theta_i), 1-\epsilon, 1+\epsilon)\hat{A}_t\right)\right)\right], \tag{2}$$

where $r^{(i)}$ is the ratio between the updated and old policy of agent i, and $\hat{A}_t$ is the advantage estimate for timestep t. In (Yu et al., 2021), generalized advantage estiamtion was used to compute $\hat{A}_t$, which combines group agent rewards and value function estimates according to

$$\begin{aligned}
&\hat{A}_t = \delta_t + (\gamma\lambda)\delta_{t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{T-1}, \\
&\text{where} \quad \delta_t = \left(\sum_{j=1}^{M} r^{(j)}_t\right) + \gamma V(s_{t+1}) - V(s_t).
\end{aligned}$$

We modify the objective in (2) by replacing advantage terms with *individual agent* advantage terms, which ignore the rewards of irrelevant agents. The objective for agent i becomes

$$L^{(i)}_{\text{PRD}} = \hat{\mathbb{E}}\left[\min\left(\left(r^{(i)}_t(\theta_i)\hat{A}_{i,t}\right), \left(\text{clip}(r^{(i)}_t(\theta_i), 1-\epsilon, 1+\epsilon)\hat{A}_{i,t}\right)\right)\right],$$

where

$$\begin{aligned}
&\hat{A}_{i,t} = \delta_{i,t} + (\gamma\lambda)\delta_{i,t+1} + ... + (\gamma\lambda)^{T-t+1}\delta_{i,T-1}, \\
&\delta_{i,t} = \left(\sum_{j \in R_i(s_t)} r^{(j)}_t\right) + \gamma V^{\psi}_i(s_{t+1}, a^{\neq i}_{t+1}) - V^{\psi}_i(s_t, a^{\neq i}_t).
\end{aligned}$$

Note in the above equation that the reward terms for agents not in $R_i(s_t)$ have been removed, and V has been replaced by the value function V^{ψ}_i described in Sec. 3.1., which is regressed against the sum of returns of agents in $R_i(s_t)$. Pseudocode for PRD-MAPPO is included in Sec. B of the appendix.

We additionally propose a "soft" variant of PRD-MAPPO, which we refer to as PRD-MAPPO-soft, that softly reweights agent rewards according to attention weights of the Q network, *i.e.*, $\delta_{i,t} = \left(\sum_{j=1}^{M} w_{ji}(s_t) r^{(j)}_t\right) + \gamma V^{\psi}_i(s_{t+1}, a^{\neq i}_{t+1}) - V^{\psi}_i(s_t, a^{\neq i}_t)$. In this soft variant, V^{ψ}_i is regressed against the *weighted sum* of agent returns, $\sum_{j=1}^{M} w_{ji}(s_t) R^{(j)}_t$.

3.3 Partial Reward Decoupling for environments with shared rewards

One drawback to our PRD approach is that it assumes individual reward streams for each agent are available, *i.e.,* at each timestep, the environment provides a separate scalar reward for each agent. However, some multi-agent systems only provide a single scalar *shared* reward for the entire group at each timestep. To deal with the shared reward setting, we propose strategy for decomposing shared returns into individual agent returns, to which we can then apply PRD. We start by training a *shared* Q function to predict the shared returns (*i.e.,* the sum of future shared rewards). Here we use a similar architecture as described in Sec. 3.1, with the one difference that our network has 1 output rather than M outputs. We denote the vector of attention weights assigned by all agents to the action of agent j as $W_{:j}$. There is one such vector for each timestep and each agent; we omit the timestep subscripting for brevity. As a heuristic to measure the overall influence that each agent j has on the future shared reward, we aggregate the attention weights for each agent j by taking the mean of $W_{:j}$, which we refer to as $\tilde{W}_j$. The individual returns for each agent j at each timestep are then set proportionally to $\tilde{W}_j$, such that they sum to the original shared return. Subsequently, we apply PRD-MAPPO to these individual returns as we would in the individual reward setting described in Sec. 3.2. We refer to this approach as PRD-MAPPO-shared.

4 Experiments

We experimentally compare the performance of the following algorithms on several cooperative MARL environments:

PRD-MAPPO (ours): MAPPO with PRD, as described in Sec. 3.1.

PRD-MAPPO-soft (ours) : the soft variant of PRD-MAPPO as described in Sec. 3.1.

PRD-MAPPO-shared (ours) : the soft variant of PRD-MAPPO in the shared reward setting, as described in Sec. 3.3.

MAPPO: a multi-agent variant of PPO, proposed by Yu et al. (2021).

HAPPO: a recent state-of-the-art algorithm proposed by Kuba et al. (2021) that extends trust region learning to cooperative multi-agent reinforcement learning (MARL), enabling monotonic policy improvement without the need for shared policy parameters.

G2ANet-MAPPO: MAPPO with a G2ANet-style critic. This baseline attempts to import the credit assignment benefits of G2ANet (which was originally used in the Actor-Critic algorithm) to the more state-of-the-art MAPPO.

Counterfactual Multi-Agent Policy Gradient (COMA): Proposed by Foerster et al. (2018), COMA is a multi-agent actor-critic method. COMA addresses credit assignment by using a counterfactual baseline that marginalizes out a single agent's action, while keeping the other agents' actions fixed, allowing COMA to better isolate each agent's contribution to group reward.

PRD-V-MAPPO: PRD-MAPPO, using the value-function-based method of relevant set estimation, as described by Freed et al. (2022). This version uses a learned value function for both relevant set and advantage estimation, and scales quadratically in time complexity with number of agents. We include this as a baseline to assess the effect of critic choice.

Learning Implicit Credit Assignment (LICA): proposed by Zhou et al. (2020b), LICA is a method for implicit credit assignment that is closely related to value gradient methods, which seek to optimize policies in the direction of approximate value gradients. LICA extends the concept of value mixing present for credit assignment found in QMix and Value-decomposition Networks by introducing an additional latent state representation into the policy gradients. The authors claim that this additional state information provides sufficient information for learning optimal cooperative behaviors without explicit credit assignment.

QMix: proposed by Rashid et al. (2018), QMix learns a joint state-action value function, represented as a complex non-linear combination of per-agent value functions. The joint value function is structurally guaranteed to be monotonic in per-agent values, allowing agents to maximize the joint value function by greedily selecting the best actions according to their own per-agent value functions.

The policy network and critic used for advantage calculations for PRD-MAPPO, PRD-MAPPO-soft, PRD-MAPPO-shared, MAPPO, HAPPO, G2ANet-MAPPO, COMA and PRD-V-MAPPO have the same architecture and number of parameters. Because LICA and QMix depend on a particular critic architecture, we used the original architectures as described by Zhou et al. (2020a) and Rashid et al. (2018) respectively. For all environments and all algorithms, we performed a grid search over hyperparameters as described in the appendix.

We consider the following environments, with detailed descriptions of each in the appendix: Collision Avoidance, Pursuit, Pressure Plate, Level-Based Foraging, and StarCraft Multi-Agent Challenge Lite (SMAClite), specifically the 5m_vs_6m, 10m_vs_11m, and 3s5z battle scenarios.

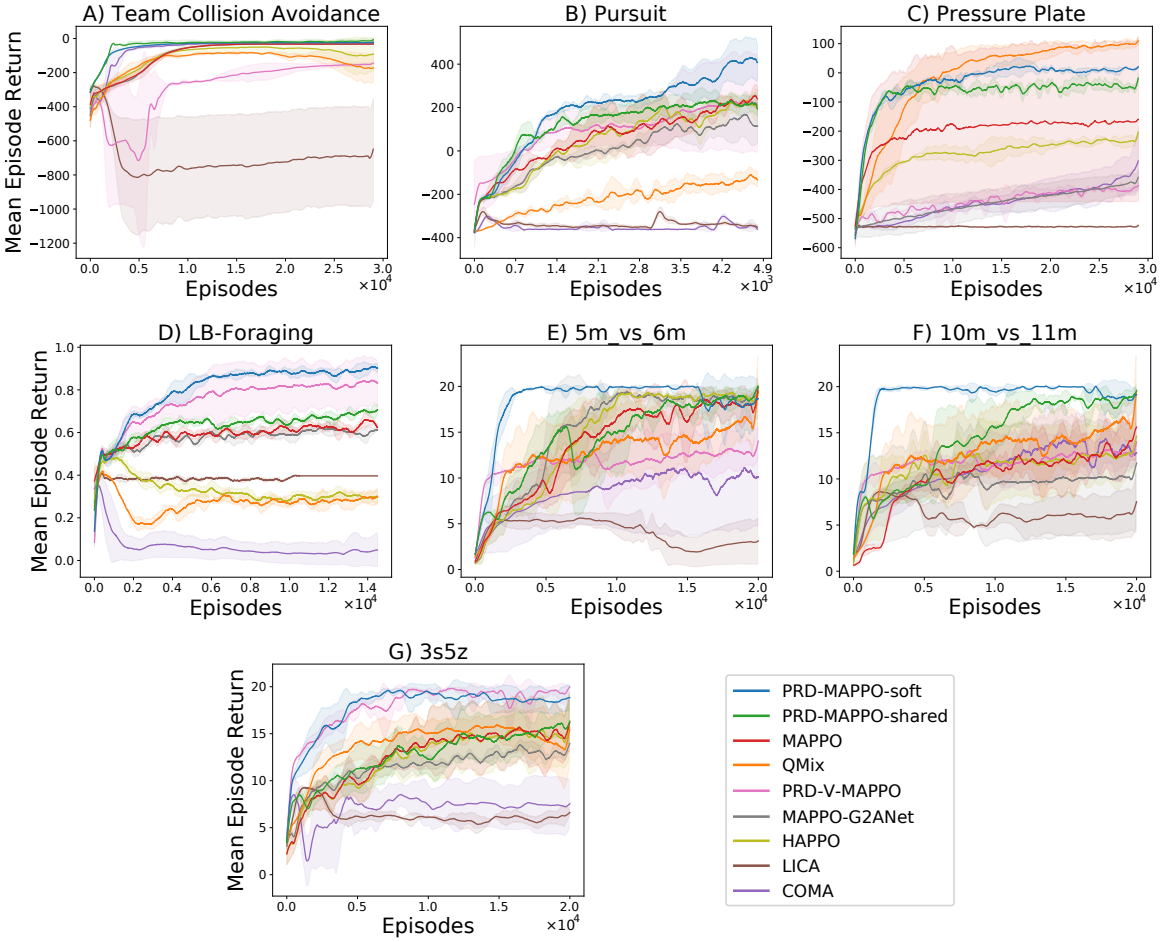

Figure 2: **Average reward vs. episode for PRD-MAPPO-soft, PRD-MAPPO, PRD-V-MAPPO, COMA, LICA, QMix, MAPPO, MAPPO-G2ANet on A) team collision avoidance, B) pursuit, C) pressure plate, D) Level-Based Foraging, E) StarCraft 5m_vs_6m, F) StarCraft 10m_vs_11m tasks, and G) StarCraft 3s5v**. Solid lines indicate the average over 5 random seeds, and shaded regions denote a 95% confidence interval. Approaches that incorporate PRD (PRD-MAPPO and PRD-MAPPO-soft) tend to outperform all other approaches, indicating that PRD can be leveraged to improve PPO by improving credit assignment.

5 Results and Discussion

The reward curves for all tasks are shown in Fig. 2. We found that of the algorithms we tested, only PRD-MAPPO-soft, PRD-MAPPO-shared, and PRD-MAPPO performed consistently well across all environments, with PRD-MAPPO-soft tending to perform the best. PRD-MAPPO-soft was outperformed only in one environment (pressure plate) by one algorithm (QMix), and in general outperformed all other algorithms on all tasks.

5.1 Relevant Set Visualization

To gain more insight into the relevant set selection process, in Fig. 3 we visualized the attention weights inferred by a trained group of agents in the Collision Avoidance task. In this task, agents are rewarded for reaching an assigned goal location while avoiding collisions. Agents are divided into three teams, consisting of agents 1-8, 9-16, and 17-24, and are only penalized for colliding with other agents on their team. We therefore expect agents to assign large attention weights only to other agents on their same team, because each agents' reward is independent of the actions of agents on other teams. Fig. 3 displays the average attention weights as an M x M grid, with the ith row and jth column corresponding to the average attention weight that agent i assigns to agent j. Because agents always assign an attention weight of 1 to themselves, we remove these elements from the visualization as they are uninformative. We find that, as expected, agents assign considerably non-zero attention weights only to other agents on their same team, while assigning near-zero attention weights to all other agents. Attention weights were averaged over 5000 independent episodes.

5.2 Policy Gradient Estimator Variance Analysis

To empirically verify the claim that partial reward decoupling decreases the variance of MAPPO policy gradient estimates, we estimate the variance of MAPPO and PRD-MAPPO at various points during training. For maximum comparability, we compute the variance for both MAPPO and PRD-MAPPO using data gathered from the same policy, taken at 1000-episode intervals during the training of PRD-MAPPO. Using these policies, we collect 100 independent batches of data, and differentiate the MAPPO or PRD-MAPPO surrogate objective evaluated on each batch, to obtain 100 independent gradient estimates for both approaches for each policy. Finally, we arrive at a scalar empirical variance estimate, by taking the trace of the covariance matrix estimated using each batch of 100 gradient estimates, along with a 95% confidence interval. The results are plotted in Fig. 4. In general, we find that PRD-MAPPO tends to avoid the spikes in gradient variance present in MAPPO, which may explain its improved stability and asymptotic performance.

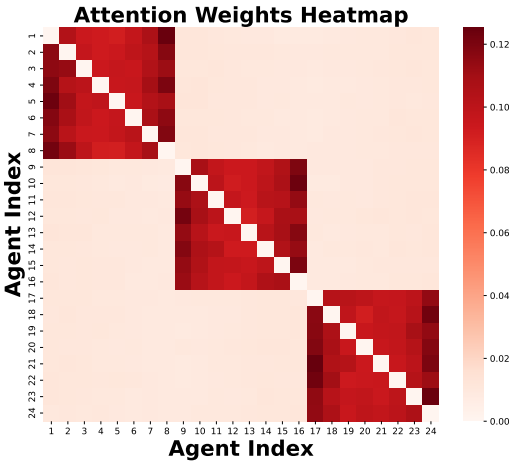

Figure 3: **Relevant set visualization in Collision Avoidance environment.** We visualize the average attention weight that each agent assigns to every other agent, averaged across 5000 independent episodes. Because agents always assign an attention weight of 1 to themselves, we remove those elements from the plot as they are uninformative. We notice that generally agents assign a far higher attention weight to agents in their team, compared to agents on other teams, which is to be expected given that only an agent's teammates are capable of influencing its rewards.

6 Related Work

Many recent approaches have been proposed to deal with the credit assignment problem. G2ANet (Liu et al., 2020), for instance, proposed a novel attention-based game abstraction mechanism that enables the critic to better isolate important interactions among

agents, and ignore unimportant ones (although explicit decoupling is not done, as in PRD). Counterfactual Multi-Agent Policy Gradient (COMA) (Foerster et al., 2018) proposed a novel *counterfactual baseline* that allows each agent to more precisely determine the effect that its action had on group reward by conditioning on the actions of all other agents. COMA builds on the idea of difference rewards (Wolpert & Tumer, 2002), in which each agent uses a modified reward that compares the shared reward to a counterfactual situation in which the agent took some *default action*. Value-decomposition actor-critics (VDAC) (Su et al., 2021) uses value decomposition networks (Sunehag et al., 2017; Rashid et al., 2018) as critics for credit assignment in the actor-critic framework. Off-policy multi-agent decomposed policy gradients (Wang et al., 2020) is another multi-agent policy-gradient algorithm that uses the idea of value decomposition, but applies it to a DDPG-style off-policy policy gradient (Silver et al., 2014). Finally, Learning Implicit Credit Assignment for Cooperative Multi-Agent Reinforcement Learning (LICA) (Zhou et al., 2020a) implicitly addressed the credit assignment problem by representing a centralized critic as a hypernetwork, and finding an end-to-end differentiable optimization setting where the policies simultaneously improve along the joint action value gradients, thus serving as a proxy for finding optimal credit assignment strategies.

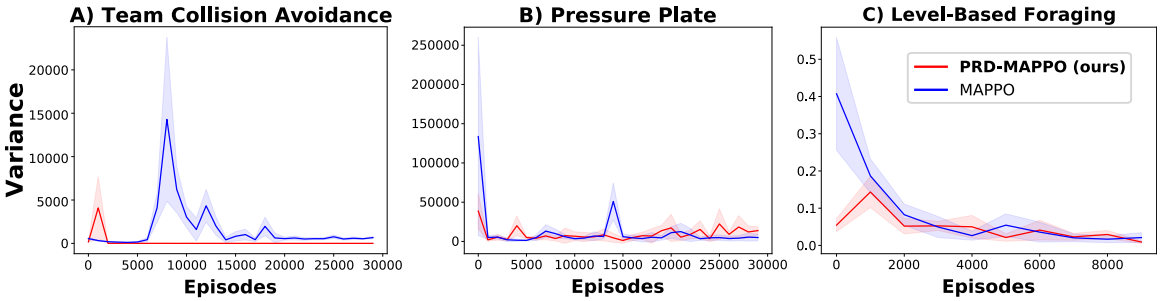

Figure 4: **Gradient estimator variance vs. episode for team collision avoidance, pressure plate, and LBF environments**. Solid lines indicate the average over 5 random seeds, and shaded regions denote a 95% confidence interval. PRD-MAPPO tends to avoid the dramatic spikes in gradient variance demonstrated by MAPPO.

7 Limitations

The primary limitation of PRD-MAPPO is that PRD is not guaranteed to accelerate learning in every environment, because some tasks cannot be decomposed (*i.e.*, each agent's relevant set contains most or all other agents). For example, in the traffic junction experiment, it is possible that learning is only somewhat improved by PRD because interactions among agents are too dense, making decoupling less effective.

8 Conclusions

We addressed the shortcomings of MAPPO, a state-of-the-art multi-agent reinforcement learning algorithm. Specifically, we hypothesized that the credit assignment problem manifests itself in policy gradient estimator variance. Based on this hypothesis, we proposed integrating PRD into MAPPO as a strategy to improve credit assignment, yielding a new multi-agent model-free RL algorithm, PRD-MAPPO. We demonstrated that PRD-MAPPO provides significant improvements both in learning efficiency and stability, across a diverse set of tasks, compared to both MAPPO and several state-of-the-art MARL algorithms such as QMix, LICA, and COMA. We empirically verified the hypothesis that PRD decreases the variance of the gradient estimates of MAPPO. Finally, we visualized the relevant sets inferred by PRD, and found that it correctly grouped together agents that should cooperate. The improvements in learning speed and stability, combined with decreased gradient variance and sensible relevant set estimation, indicate that PRD, used in the context of MAPPO, provides a useful credit assignment strategy for multi-agent problems.

References

Gabriel Barth-Maron, Matthew W Hoffman, David Budden, Will Dabney, Dan Horgan, TB Dhruva, Alistair Muldal, Nicolas Heess, and Timothy Lillicrap. Distributed distributional deterministic policy gradients. In *International Conference on Learning Representations*, 2018.

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

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.

Benjamin Freed, Aditya Kapoor, Ian Abraham, Jeff Schneider, and Howie Choset. Learning cooperative multi-agent policies with partial reward decoupling. *IEEE Robotics and Automation Letters*, 7(2):890–897, 2022. doi: 10.1109/LRA.2021.3135930.

Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer. Cooperative multi-agent control using deep reinforcement learning. In *International Conference on Autonomous Agents and Multiagent Systems*, pp. 66–83. Springer, 2017.

Max Jaderberg, Wojciech M Czarnecki, Iain Dunning, Luke Marris, Guy Lever, Antonio Garcia Castaneda, Charles Beattie, Neil C Rabinowitz, Ari S Morcos, Avraham Ruderman, et al. Human-level performance in 3d multiplayer games with population-based reinforcement learning. *Science*, 364(6443):859–865, 2019.

Vijay R Konda and John N Tsitsiklis. Actor-critic algorithms. In *Advances in neural information processing systems*, pp. 1008–1014, 2000.

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. In *International Conference on Learning Representations*, 2021.

Michael L Littman. Markov games as a framework for multi-agent reinforcement learning. In *Machine learning proceedings 1994*, pp. 157–163. Elsevier, 1994.

Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. Multi-agent game abstraction via graph attention neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 7211–7218, 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.

Marvin Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

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*, pp. 1928–1937. PMLR, 2016.

Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 4295–4304. PMLR, 2018.

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

John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*, 2015b.

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

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

Jianyu Su, Stephen Adams, and Peter Beling. Value-decomposition multi-agent actor-critics. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 11352–11360, 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.

Richard S Sutton, Andrew G Barto, et al. *Introduction to reinforcement learning*, volume 135. MIT press Cambridge, 1998.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 34: 15032–15043, 2021.

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

Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. Off-policy multi-agent decomposed policy gradients. *arXiv preprint arXiv:2007.12322*, 2020.

David H Wolpert and Kagan Tumer. Optimal payoff functions for members of collectives. In *Modeling complexity in economic and social systems*, pp. 355–369. World Scientific, 2002.

Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.

Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 33:11853–11864, 2020a.

Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2007.02529*, 2020b.

A Detail Task Descriptions

Collision Avoidance: 3 teams of 8 agents each exist in a square bounded 2D region. Agents receive a reward for reaching their assigned goal location, and receive a penalty for colliding with other agents belonging to the same team. Agents therefore need only cooperate with other agents on their team to avoid collisions. Both the agents and goals are initialized in random locations. The observation space consists of the agent's position, velocity, team ID, and goal position. The agents can take 5 possible actions that allow them to move either north, south, east, west or remain stationary. The reward function is the l2 distance between the agent position and the goal position multiplied by a scalar value of 0.1. On collision, the participating agents receive a -1 reward each. The environment

terminates if all agents reach their assigned goal location or 100 timesteps run out. While training decentralized policies, relative positions of all other agents and their team ID are also included in the observation space. Episodes last a maximum of 100 timesteps. This environment was modified from the cooperative navigation environment first developed by Lowe et al. (2017). The code for this environment can be found at `https://github.com/openai/multiagent-particle-envs` (MIT License).

Pursuit: 8 agents exist in a 16 x 16 grid with an obstacle in the center. To receive a reward, two agents must coordinate their actions to surround randomly moving "evader" agents on two sides. There are 30 evaders in the environment. Each pursuer observes a 7 x 7 grid centered around itself with 3 channels, indicating the positions of walls, other agents, and evaders, respectively. Once an evader is caught, it is removed from the environment. The environment terminates when every evader has been caught, or when 500 timesteps are completed. The environment is available in the PettingZoo MARL benchmark suite (Terry et al., 2021) at `https://pettingzoo.farama.org/environments/sisl/pursuit/` (MIT License) and was first proposed by Gupta et al. (2017).

Pressure plate: 6 agents exist in a grid, divided into 6 separate chambers by gates. In any given chamber, a particular agent can open the gate by standing on a special grid cell known as the pressure plate. To successfully solve the task, this agent must remain on the pressure plate until the other agents have successfully moved into the next chamber. The goal is for one particular agent to traverse all six chambers and arrive at a goal location in the final chamber. Each agent observes a 5x5 square around its location, with a separate channel for each type of entity in the environment (e.g., walls, pressure plates, doors, agents, and goals). The agent's (x,y) coordinates are concatenated to the end of the observation vector. The action space is discrete and has five possibilities: up, down, left, right, and remain stationary. Each agent receives rewards independent of other agents. If an agent is in the room that contains their assigned plate, their reward is the negative normalized Manhattan distance between their current position and the plate. Otherwise, their reward is the number of rooms between their current room and the room that contains their assigned plate. Episodes last a maximum of 70 timesteps. The code for this environment is available at `https://github.com/uoe-agents/pressureplate` (MIT License).

Level-Based Foraging (LBF): Agents navigate a grid world and collect food items by cooperating with other agents. Each agent and food item is assigned a level and are randomly distributed throughout the environment. Successfully collecting a food item of a particular level requires the sum of the levels of the agents involved to be greater than or equal to the level of the food item. Agents are rewarded based on the level of the food items they help collect, divided by their contribution (their level). Reward discounting incentivizes agents to collect all food items as quickly as possible to maximize returns. The observation space consists of the agent's position in the grid, its level, relative positions of all other agents and food items, and their levels. The agents can either move in one of the four directions, collect a food item, or do nothing. Episodes last a maximum of 70 timesteps. The code for this environment can be found at `https://github.com/semitable/lb-foraging` (MIT License).

Lightweight StarCraft (SMAClite): SMAClite is a lightweight version of the StarCraft II game engine. It is computationally less expensive relative to SC II and provides a simple "pythonic" framework to add custom environments and make alterations to the environment logic. The observation space consists of the relative positions, unit type, health and shield strength of the agent's allies and enemies within the field of view of the agent and the health and shield strength of itself. The agents can move in any of the 4 cardinal directions, remain stationary, or attack any of the enemy agent within its field of view. Each combat scenario is run for 100 timesteps, though agents may die before this time. We consider three different battle scenarios, 1) 5m_vs_6m, where 5 agent-controlled marines battle 6 enemy marines, 2) 10m_vs_11m, where 10 agent-controlled marines battle 11 enemy marines, and 3) 3s5z, where 3 agent-controlled stalkers and 5 agent-controlled zealots battle 3 enemy stalkers and 5 enemy zealots. The code for SMAClite is available at `https://github.com/uoe-agents/smaclite` (MIT License).

B Pseudocode

Algorithm 1 PRD-MAPPO

Initialize θ, the parameters for policy π, ω, the parameters for state-action value critic Q and ϕ, the parameters for state value critic V, using orthogonal initialization (Hu et al., 2020)
Set learning rate α
while step $\leq step_{\max}$ **do**
  set data buffer $D = \{\}$
  for $i = 1$ to *batch_size* **do**
    $\tau = []$ – empty list
    initialize $h^{(1)}_{0,\pi}, \ldots, h^{(M)}_{0,\pi}$ actor RNN states
    initialize $h^{(1)}_{0,V}, \ldots, h^{(M)}_{0,V}$ state value RNN states
    initialize $h^{(1)}_{0,Q}, \ldots, h^{(M)}_{0,Q}$ state-action value RNN states
    for $t = 1$ to T **do**
      for all agents a **do**
        $u^{(a)}_t, h^{(a)}_{t,\pi} = \pi(o^{(a)}_t, h^{(a)}_{t-1,\pi}; \theta)$
      end for
      $(q^{(1)}_t, \ldots q^{(M)}_t), (h^{(1)}_{t,Q} \ldots h^{(M)}_{t,Q}), W_{\text{prd},t} = Q(s^{(1)}_t \ldots s^{(M)}_t, u^{(1)}_t \ldots u^{(M)}_t, h^{(1)}_{t-1,Q} \ldots h^{(M)}_{t-1,Q}; \omega)$
      $(v^{(1)}_t, \ldots v^{(M)}_t), (h^{(1)}_{t,V} \ldots h^{(M)}_{t,V}) = V(s^{(1)}_t \ldots s^{(M)}_t, u^{(1)}_t \ldots u^{(M)}_t, h^{(1)}_{t-1,V} \ldots h^{(M)}_{t-1,V}; \phi)$ – we mask out the actions of agent a while calculating its state value $v^{(a)}$
      Execute actions u_t, observe r_t, s_{t+1}, o_{t+1}
      τ += $[s_t, o_t, h_{t,\pi}, h_{t,V}, u_t, r_t, s_{t+1}, o_{t+1}]$
    end for
    Compute relevant set $R_1, ..., R_M$ using W_{prd}
    Compute return G_i for each agent $i = 1, ..., M$, to learn the Q function and total relevant-set return $\bar{G}_i = \sum_{j \in R_i} G_j$ for each agent i to learn V function on τ and normalize with PopArt
    Compute advantage estimate $\hat{A}^1, ..., \hat{A}^M$ via GAE on state value estimates on τ, using PopArt
    Split trajectory τ into chunks of length L
    for $l = 0, 1, \ldots, T//L$ **do**
      $D = D \cup (\tau[l : l + T], \hat{A}[l : l + L], G[l : l + L], \bar{G}[l : l + L])$
    end for
  end for
  for mini-batch $k = 1, \ldots, K$ **do**
    $b \leftarrow$ random mini-batch from D with all agent data
    for each data chunk c in the mini-batch b **do**
      update RNN hidden states for π, Q and V from first hidden state in data chunk
    end for
  end for
  Adam update θ on $L(\theta)$ with data b
  Adam update ω on $L(\omega)$ with data b
  Adam update ϕ on $L(\phi)$ with data b
end while

C Additional Results

We experimented with various methods for selecting agent relevant sets, as described below. Reward curves for each method in each of our four environments is shown in Fig. 5. **PRD-MAPPO**: As described in Sec. 3.1 of the manuscript. The attention-weight threshold ϵ used to agent relevant sets is held constant through training.

PRD-MAPPO-soft: As described in Sec. 4 of the manuscript. A variant of PRD-MAPPO in which advantage terms are not excluded from the PPO update according to hard thresholding, but rather advantage terms for each agent i are softly re-weighted according to the attention weights applied by other agents to the actions of agent i.

PRD-MAPPO-ascend: Attention-weight threshold ϵ is linearly increased from 0 to θ over the first N policy updates and then held constant, where θ and N are hyperparameters. This method transitions from including all agents in the relevant set to having only a subset of agents in the relevant set.

PRD-MAPPO-decay: Attention-weight threshold ϵ is linearly decreased from *theta* to 0 over the first N policy updates, and then held constant. In this case, agents aggressively prune relevant sets early on, transitioning to standard MAPPO by the end of training.

PRD-MAPPO-G2ANet: A semi-hard attention mechanism based on G2ANet Liu et al. (2020) is used to select relevant sets. Agents are excluded from the relevant set if their associated attention weight is exactly 0. This approach has the advantage that it allows a manual threshold on attention weights to be avoided.

PRD-MAPPO-top-k: The agents with the top k highest attention weights are included in the relevant set (where k is a hyperparameter).

D Implementation Details

The code was run on Lambda Labs deep learning workstation with 2-4 Nvidia RTX 2080 Ti graphics cards. Each training run was run on one single GPU, and required approximately 2 days. The hyperparamers used for our experiments are reported in the tables below:

E Hyperparameters

Hyperparameters used for MAPPO variants, PRD variants, PRD_V_MAPPO, QMix, LICA and COMA that are common to all tasks are shown in Tables 23, 4 5, and 6 respectively. The task-specific hyperparameters considered in our grid search for MAPPO variants, PRD variants, PRD_V_MAPPO QMix, LICA, and COMA are shown in Tables 7, 8, 9 10, 11, and 12, respectively. Bold values indicate the optimal hyperparameters.

Table 1: Episodic Length of all environments

common environment	**max timesteps**
collision avoidance	100
pursuit	500
pressure plate	70
level-based foraging	70
5m_vs_6m	100
10m_vs_11m	100
3s5z	100

Table 2: Common Hyperparameters for all algorithms in all domains

common hyperparameters	**value**
optimizer	AdamW
gamma	0.99
gae lambda	0.95
weight decay	0.0
optim epsilon	1e-5
max grad norm	10.0
network initialization	orthogonal

Table 3: Common Hyperparameters for MAPPO, HAPPO, MAPPO-G2ANet, PRD-V-MAPPO, PRD-MAPPO-shared and PRD-MAPPO-soft algorithms in all domains

common hyperparameters	**value**
critic loss	huber loss
huber delta	10.0
num mini-batch	1
gae lambda	0.95
actor network	rnn
recurrent data chunk length	10
recurrent num layers	1
rnn hidden dim	64
value normalization	PopArt

Table 4: Common Hyperparameters for QMix in all domains.

common hyperparameters	**value**
buffer size	5000
batch size	32
hypernet layers	2
hypernet hidden dim	32
target network update interval	200
td lambda	0.8
epsilon decay steps	2000 episodes
epsilon start	1.0
epsilon end	0.1
value loss	huber loss
huber delta	10.0
q network	rnn
rnn hidden dim	64
recurrent data chunk length	10
recurrent num layers	1

Table 5: Common Hyperparameters for LICA.

common hyperparameters	**value**
hypernet layers	2
hypernet hidden dim	64
target network update interval	200
td lambda	0.8
critic loss	huber loss
huber delta	10.0
actor network	rnn
actor rnn hidden dim	64
actor recurrent data chunk length	10
actor recurrent num layers	1

Table 6: Common Hyperparameters for COMA.

common hyperparameters	**value**
target network update interval	200
td lambda	0.8
critic loss	huber loss
huber delta	10.0
actor network	rnn
rnn hidden dim	64
recurrent data chunk length	10
recurrent num layers	1

Table 7: MAPPO and MAPPO-G2ANet hyperparameter sweep. Bold values indicate the optimal hyperparameters.

Environment Name	epochs	num_episodes	value_lr	policy_lr	clip	entropy_pen
Collision Avoidance	[**5**, 10, 15]	[5, **10**]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[**0.05**, 0.2]	[1e-3, **8e-3**, 1e-2]
Pursuit	[**5**, 10, 15]	[2, **5**, 10]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[0.05, **0.2**]	[1e-3, 8e-3, **1e-2**]
Pressure Plate	[**5**, 10, 15]	[5, 7, **10**]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[0.05, 0.1, **0.2**]	[1e-3, **1e-2**, 5e-2, 1e-1]
Level-Based Foraging	[1, **5**, 10]	[1, 5, **10**]	[**5e-4**, 1e-3, 5e-3]	[**5e-4**, 1e-3, 5e-3]	[0.1, **0.2**]	[1e-3, 5e-3, **1e-2**]
5m_vs_6m	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, 5e-3, **1e-2**]
10m_vs_11m	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, 5e-3, **1e-2**]
3s5z	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, 5e-3, **1e-2**]

Table 8: PRD-MAPPO-global and PRD-MAPPO-soft hyperparameter sweep. Bold values indicate the optimal hyperparameters.

Environment Name	epochs	num_episodes	value_lr	policy_lr	clip	entropy_pen
Collision Avoidance	[**5**, 10, 15]	[5, **10**]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[**0.05**, 0.2]	[0.0, **1e-3**, 8e-3]
Pursuit	[**5**, 10, 15]	[2, **5**]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[0.05, **0.2**]	[**1e-3**, 8e-3, 1e-2]
Pressure Plate	[**5**, 10, 15]	[5, 7, **10**]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[0.1, **0.2**]	[**1e-3**, 1e-2, 5e-2, 1e-1]
Level-Based Foraging	[1, **5**, 10]	[1, 5, **10**]	[**5e-4**, 1e-3, 5e-3]	[**5e-4**, 1e-3, 5e-3]	[0.1, **0.2**]	[0.0, **1e-3**, 8e-3]
5m_vs_6m	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, **1e-3**, 1e-2]
10m_vs_11m	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, **1e-3**, 1e-2]
3s5z	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, **1e-3**, 1e-2]

Table 9: PRD-V-MAPPO hyperparameter sweep. Bold values indicate the optimal hyperparameters.

Environment Name	epochs	num_episodes	value_lr	policy_lr	clip	entropy_pen	threshold
Collision Avoidance	[**5**, 10, 15]	[5, **10**]	[1e-4, **5e-4**, 1e-3]	[1e-4, **5e-4**, 1e-3]	[**0.05**, 0.2]	[0.0, 1e-3, **1e-2**]	[0.05, **0.12**, 0.2]
Pursuit	[**5**, 10, 15]	[2, **5**]	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[0.05, **0.2**]	[1e-3, 8e-3, **1e-2**]	[0.2, **0.3**, 0.5]
Pressure Plate	[**5**, 10, 15]	[5, 7, **10**]	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[0.1, **0.2**]	[1e-3, 1e-2, **5e-2**, 1e-1]	[**0.2**, 0.4]
Level-Based Foraging	[1, **5**, 10]	[1, 5, **10**]	[**5e-4**, 1e-3, 5e-3]	[**5e-4**, 1e-3, 5e-3]	[0.1, **0.2**]	[0.0, 1e-3, **8e-3**]	[0.15, **0.2**, 0.33]
5m_vs_6m	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, 5e-3, **1e-2**]	[0.15, **0.2**, 0.33]
10m_vs_11m	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, 5e-3, **1e-2**]	[**0.1**, 0.2, 0.33]
3s5z	[1, **5**, 10]	[5, **10**]	[1e-4, 3e-4, **5e-4**]	[1e-4, 3e-4, **5e-4**]	[0.1, **0.2**]	[0.0, 5e-3, **1e-2**]	[**0.12**, 0.2, 0.33]

Table 10: Hyperparameter sweep for QMix. Bold values were selected for training the agent.

Environment Name	learning rate	update interval (episodes)	hard interval
Collision Avoidance	[1e-4, **5e-4**, 1e-3]	[5, **10**, 20]	[100, **200**, 500]
Pursuit	[1e-4, **5e-4**, 1e-3]	[**5**, 10, 20]	[100, **200**, 500]
Pressure Plate	[1e-4, **5e-4**, 1e-3]	[5, **10**, 20]	[100, **200**, 500]
LB-Foraging	[1e-4, **5e-4**, 1e-3]	[5, **10**, 20]	[100, **200**, 500]
5m_vs_6m	[1e-4, **5e-4**, 1e-3]	[5, **10**, 20]	[100, **200**, 500]
10m_vs_11m	[1e-4, **5e-4**, 1e-3]	[5, **10**, 20]	[100, **200**, 500]
3s5z	[1e-4, **5e-4**, 1e-3]	[5, **10**, 20]	[100, **200**, 500]

Table 11: Hyperparameter sweep for LICA. Bold values were selected for training the agent.

Environment Name	critic_lr	actor_lr	entropy_coeff
Collision Avoidance	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-2, **1e-1**]
Pursuit	[**1e-4**, 5e-4, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-2, **1e-1**]
Pressure Plate	[**1e-4**, 5e-4, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-2, **1e-1**]
LB-Foraging	[1e-3, **5e-3**, 1e-2]	[**1e-3**, 5e-3, 1e-2]	[1e-2, **1e-1**]
5m_vs_6m	[1e-4, **5e-4**, 1e-2]	[1e-4, **5e-4**, 1e-3]	[1e-2, **1e-1**]
10m_vs_11m	[1e-4, **5e-4**, 1e-2]	[1e-4, **5e-4**, 1e-3]	[1e-2, **1e-1**]
3s5z	[1e-4, **5e-4**, 1e-2]	[1e-4, **5e-4**, 1e-3]	[1e-2, **1e-1**]

Table 12: Hyperparameter sweep for COMA. Bold values indicate the optimal hyperparameters.

Environment Name	value_lr	policy_lr	entropy_coeff
Collision Avoidance	[1e-4, 5e-4, **1e-3**]	[5e-4, **7e-4**, 1e-3]	[1e-3, 8e-3, **1e-2**]
Pursuit	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-3, **8e-3**, 1e-2]
Pressure Plate	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-3, **8e-3**, 1e-2]
LB-Foraging	[1e-3, **5e-3**, 1e-2]	[**1e-3**, 5e-3, 1e-2]	[1e-3, 8e-3, **1e-2**]
5m_vs_6m	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-3, 8e-3, **1e-2**]
10m_vs_11m	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-3, 8e-3, **1e-2**]
3s5z	[1e-4, **5e-4**, 1e-3]	[**1e-4**, 5e-4, 1e-3]	[1e-3, 8e-3, **1e-2**]

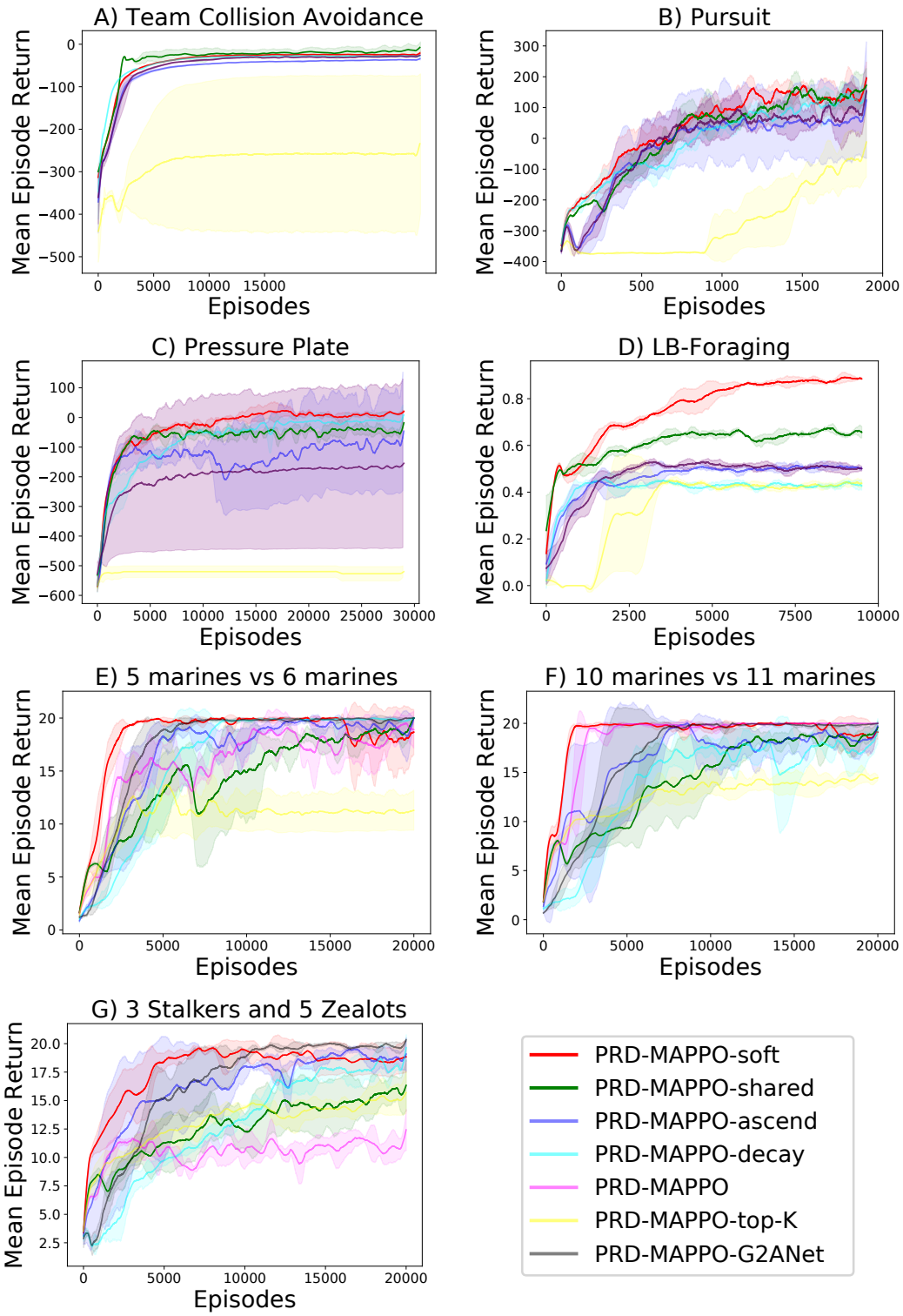

Figure 5: **Average reward vs. episode for PRD-MAPPO-soft, PRD-MAPPO-shared, PRD-MAPPO-ascend, PRD-MAPPO-decay, PRD-MAPPO, PRD-MAPPO-top-K, and PRD-MAPPO-G2ANet on A) team collision avoidance, B) pursuit, C) pressure plate, D) Level-Based Foraging tasks, E) StarCraft 5 marines vs. 6 marines, F) StarCraft 10 marines vs. 11 marines, and G) StarCraft 3 Stalkers and 5 Zealots**. Solid lines indicate the average over 5 random seeds, and shaded regions denote a +/- 1 standard deviation confidence interval. PRD-MAPPO-soft tended to perform the best across all tasks.