RPM: GENERALIZABLE BEHAVIORS FOR MULTI-AGENT REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Despite the recent advancement in multi-agent reinforcement learning (MARL), the MARL agents easily overfit the training environment and perform poorly in the evaluation scenarios where other agents behave differently. Obtaining generalizable policies for MARL agents is thus necessary but challenging mainly due to complex multi-agent interactions. In this work, we model the problem with Markov Games and propose a simple yet effective method, ranked policy memory (RPM), to collect diverse multi-agent trajectories for training MARL policies with good generalizability. The main idea of RPM is to maintain a look-up memory of policies. In particular, we try to acquire various levels of behaviors by saving policies via ranking the training episode return, *i.e.*, the episode return of agents in the training environment; when an episode starts, the learning agent can then choose a policy from the RPM as the behavior policy. This innovative self-play training framework leverages agents' past policies and guarantees the diversity of multi-agent interaction in the training data. We implement RPM on top of MARL algorithms and conduct extensive experiments on Melting Pot. It has been demonstrated that RPM enables MARL agents to interact with unseen agents in multi-agent generalization evaluation scenarios and complete given tasks, and it significantly boosts the performance up to 402% on average.

1 INTRODUCTION

Recent years have witnessed considerable progress in Multi-Agent Reinforcement Learning (MARL) research (Yang & Wang, 2020). In MARL, each agent acts decentrally and interacts with other agents to complete particular tasks or achieve specific goals via reinforcement learning (RL). However, generalization (Hupkes et al., 2020) remains a critical issue in MARL research. Generalization may have different meanings from different perspectives and research directions. Such as generalization at the representation level studied in supervised learning and generalization to different test environments rather than the training environments for a single agent (Kirk et al., 2021). In this work, we study the the generalization ability of agents to collaborate or compete with other agents with unseen policies during training. Such a setup is critical to real-world MARL applications (Leibo et al., 2021). Unfortunately, current MARL methods mostly neglect generalization issues and could be fragile; for example, a co-player changing its policy may cause the trained agent to fail to cooperate.

In this work, we aim to train MARL agents that can adapt to new scenarios where other agents' policies are unseen during training for MARL. To understand the difficulties and why it is crucial to tackle generalization, we illustrate a two-agent stag-hunt game as an example in Figure 1. The agents are trained to obtain policies maximizing the group reward by shooting arrows to the stag. As a result, they may perform well in evaluation scenarios similar to the training environment, as shown in Figure 1 (left) and (middle), respectively. However, these agents may fail when evaluated in scenarios different from the training scenarios. As shown in Figure 1 (right), the learning agent (called the focal agent following the convention in (Leibo et al., 2021)) is supposed to work together with another agent (called the background agent following the naming in (Leibo et al., 2021)) are pre-trained to be selfish (*i.e.*, only capture the hare). In this case, the focal agent will fail to capture the stag without the help from its teammates and the optimal policy to capture the hare. However, background agents are unseen to the focal agent during training. Therefore, without generalization, the agents trained as Figure 1 (left) cannot achieve an optimal policy in the new evaluation scenario.

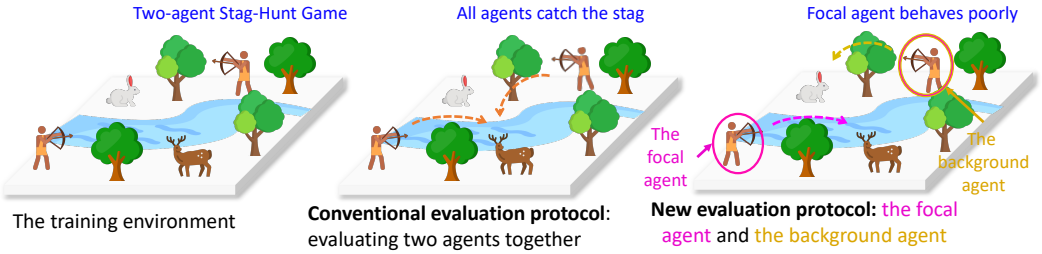


Figure 1: Two-Agent Stag-Hunt Game. **Left:** Training environment. Two agents (hunters) hunting in the environment. **Middle:** After training in the training environment, all agents behave cooperatively to capture the stag for maximizing the group reward. **Right:** In the new evaluation scenario, one agent is picked as the focal agent (in the magenta circle) and paired with a pre-trained agent (in brown circle) that behaves in different ways to evaluate the performance of the selected agent. The conventional evaluation protocol fails to evaluate such behavior and current MARL easily fails to learn the optimal policy.

We model the problem with Markov games (Littman, 1994) and propose a simple yet effective method called ranked policy memory (RPM) to attain generalizable policies in multi-agent systems during training. The core idea of RPM is to maintain a look-up memory of policies during training for the agents. In particular, we first evaluate the trained agents' policies after each training update. We then rank and save the trained agents' policies by the training episode returns. In this way, we obtain various levels, *i.e.*, performances, of policies. When starting an episode, the agent can access the memory and load the randomly sampled policy to replace the current behavior policy. The new ensemble of policies enables agents in the self-play framework to collect diversified experiences in the training environment for training. These diversified experiences contain many novel multi-agent interactions that enhance the extrapolation capacity of MARL, boosting the generalization performance. We note that an easy extension to having different behavior properties as the keys in RPM could potentially further enrich the generalization but it is left for future work.

We implement RPM on top of state-of-the-art MARL algorithm, MAPPO (Yu et al., 2021). To verify its effectiveness, we use Melting Pot (Leibo et al., 2021) as our testbeds. We then conduct large-scale experiments with the Melting Pot benchmark, which is a well-recognized benchmark for MARL generalization evaluation. The experiment results demonstrate that RPM significantly boosts the performance of generalized social behaviors up to 402% on average and outperforms many baselines in a variety of multi-agent generalization evaluation scenarios. Our code, pictorial examples, and videos are available at this link: `https://sites.google.com/view/rpm-iclr2023/`.

2 Preliminaries

2.1 Markov Games and Notations

We consider the Markov Games (Littman, 1994) represented by a tuple $\mathcal{G} = \langle \mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{O}, P, R, \gamma, \rho \rangle$. $\mathcal{N}$ is a set of agents with the size $|\mathcal{N}| = N$; $\mathcal{S}$ is a set of states; $\mathcal{A} = \times_{i=1}^{N} \mathcal{A}_i$ is a set of joint actions with $\mathcal{A}_i$ denoting the set of actions for an agent i; $\mathcal{O} = \times_{i=1}^{N} \mathcal{O}_i$ is the observation set, with $\mathcal{O}_i$ denoting the observation set of the agent i; $P : \mathcal{S} \times \mathcal{A} \to \mathcal{S}$ is the transition function and $R = \times_{i=1}^{N} r_i$ is the reward function where $r_i : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ specifies the reward for the agent i given the state and the joint action; γ is the discount factor; the initial states are determined by a distribution $\rho : \mathcal{S} \to [0, 1]$. Given a state $s \in \mathcal{S}$, each agent $i \in \mathcal{N}$ chooses its action u_i and obtains the reward $r(s, \boldsymbol{u})$ with the private observation $o_i \in \mathcal{O}_i$, where $\boldsymbol{u} = \{u_i\}_{i=1}^{N}$ is the joint action. The joint policy of agents is denoted as $\boldsymbol{\pi_\theta} = \{\pi_{\theta_i}\}_{i=1}^{N}$ where $\pi_{\theta_i} : \mathcal{S} \times \mathcal{A}_i \to [0, 1]$ is the policy for the agent i. The objective of each agent is to maximize its own total expected return $R_i = \sum_{t=0}^{\infty} \gamma^t r_i^t$.

2.2 Multi-Agent Reinforcement Learning

In MARL, multiple agents act in the environment to maximize their respective returns with RL (Sutton & Barto, 2018). Each agent's policy π_i is optimized by maximizing the following objective:

$$\mathcal{J}(\pi_i) \triangleq \mathbb{E}_{s_{0:\infty} \sim \rho_{\mathcal{G}}^{0:\infty}, a_{0:\infty}^{i} \sim \pi_i} \left[\sum_{t=0}^{\infty} \gamma^t r_t^i \right],$$

where $\mathcal{J}(\pi_i)$ is a performance measure for policy gradient RL methods (Williams, 1992; Lillicrap et al., 2016; Fujimoto et al., 2018). Each policy's Q value Q_i is optimized by minimizing the

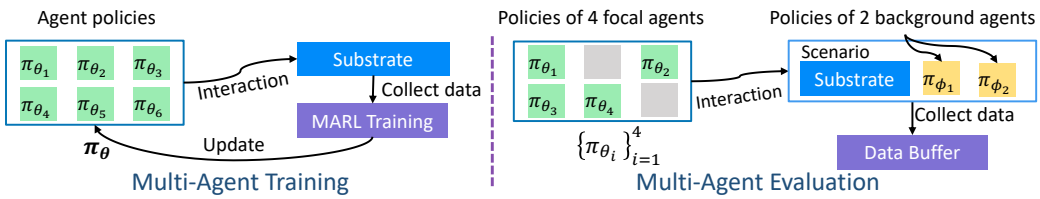


Figure 2: An example of our formulation. **Left:** All six agents' policies are trained with the MARL method. **Right:** Two agents with policies π_{ϕ_1} and π_{ϕ_2} are selected as background agents and the rest of the 4 agents (with new indices) are focal agents to be evaluated. The substrate and the background agents constitute the evaluation scenario

following regression loss (Watkins & Dayan, 1992; Mnih et al., 2015) with TD-learning:

$$\mathcal{L}(\theta_i) \triangleq \mathbb{E}_{\mathcal{D}' \sim \mathcal{D}} \left[\left(y_t^i - Q_{\theta_i}^i \left(\boldsymbol{s}_t, \boldsymbol{u}_t, s_t^i, u_t^i \right) \right)^2 \right],$$

where $y_t^i = r_t^i + \gamma \max_{\boldsymbol{u}'} Q_{\bar{\theta}_i}^i \left(\boldsymbol{s}_{t+1}, \boldsymbol{u}', s_t^i, u^{i,\prime} \right)$. θ_i are the parameters of the agents. $\bar{\theta}_i$ is the parameter of the target Q^i and periodically copied from θ. $\mathcal{D}'$ is a sample from the replay buffer $\mathcal{D}$.

3 Problem Formulation

We introduce the formulation of MARL for training and evaluation in our problem. Our goal is to improve generalizabiliby of MARL policies in scenarios where policies of agents or opponents are unseen during training while the physical environment is unchanged. Following Leibo et al. (2021), the training environment is defined as *substrate*. Each substrate is an N-agent partially observable Markov game $\mathcal{G}$. Each agent optimizes its policy π_{θ_i} via the following protocol.

Definition 1 (Multi-Agent Training)**.** There are N agents act in the substrate, which is denoted as $\mathcal{G}$. Each agent receives partial environmental observation not known to other agents and aims to optimizes its policy π_{θ_i} by optimizing its accumulated rewards: $\sum_{t=0}^{\infty} \gamma^t r_t^i$. The performance of the joint policy $\boldsymbol{\pi_\theta} = \{\pi_{\theta_i}\}_{i=1}^N$ is measured by the mean individual return: $\bar{R}(\boldsymbol{\pi_\theta}) = \frac{1}{N} \sum_{i=1}^N R(\pi_{\theta_i}; \mathcal{G})$. $R(\pi_{\theta_i}; \mathcal{G})$ measures the episode return of policy π_{θ_i} in game $\mathcal{G}$ for agent i.

In order to evaluate the trained MARL policies in evaluation scenario $\mathcal{G}'$, we follow the evaluation protocol defined by Leibo et al. (2021):

Definition 2 (Multi-Agent Evaluation)**.** There are M $(1 \leq M \leq N-1)$ focal agents that are selected from N agents. The focal agents are agents to be evaluated in evaluation scenarios. They are paired with $N-M$ background agents whose policies $\boldsymbol{\pi_\phi} = \{\pi_{\phi_j}\}_{j=1}^{N-M}$ were pre-trained with pseudo rewards in the same physical environment where the policies $\boldsymbol{\pi_\theta}$ are trained. To measure the generalized performance in evaluation scenarios, we use the mean individual return of focal agents as the performance measure: $\bar{R}(\{\pi_\theta\}_{i=1}^M) = \frac{1}{M} \sum_{i=1}^M R(\pi_{\theta_i}; \mathcal{G}')$.

We show an example of our formulation in Figure 2. Note that the focal agents cannot utilise the interaction data collected during evaluation to train or finetune their policies. Without training the policies of focal agents with the collected trajectories during evaluation, the focal agents should behave adaptively to interact with the background agents to complete challenging multi-agent tasks. It is also worth noting that the ad-hoc team building (Stone & Kraus, 2010; Gu et al., 2021) is different from our formulation both in the training and evaluation. We discuss the differences in the related works section (Paragraph 3, Section 6).

4 Methodology

We propose a Ranked Policy Memory (RPM) method to provide diversified multi-agent behaviors for self-play to improve generalization of MARL. Then, we incorporate RPM into MAPPO (Yu et al., 2021) for training MARL policies.

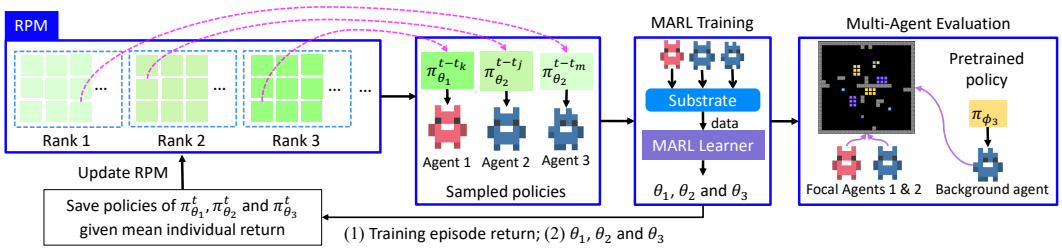


Figure 3: The workflow of RPM for a three-agent substrate. In the workflow, there are 3 agents in the substrate. Agent 3 is the background agent. Agents 1 and 2 are focal agents.

4.1 RPM: Ranked Policy Memory

In MARL, the focal agents need adaptively interact with background agents to complete given tasks. Formally, we define the objective for optimizing performance of the focal agents without exploiting their trajectories in the evaluation scenario for training the policies $\{\pi_{\theta_j}\}_{j=1}^M$:

$$\max \mathcal{J}(\{\pi_{\theta_j}\}_{j=1}^M) \triangleq \max \mathbb{E}_{s_{0:\infty}\sim\rho^{0:\infty}_{\mathcal{G}'}, a^j_{0:\infty}\sim\{\pi_{\theta_j}\}_{j=1}^M} \left[\sum_{t=0}^{\infty} \gamma^t \frac{1}{M} \sum_{j=1}^{M} r^j_t \middle| \mathcal{G}' \right]. \tag{1}$$

To improve the generalization performance of MARL, it is crucial for agents in the substrate to cover as much as multi-agent interactions, *i.e.*, data, that resemble the unseen multi-agent interactions in the evaluation scenario. However, current training paradigms, like independent learning (Tampuu et al., 2017) and centralized training and decentralized execution (CTDE) (Oliehoek et al., 2008), cannot give diversified multi-agent interactions, as the agents' policies are trained at the same pace. To address this issue, we propose to gather massive diversified agent-agent interaction data for multi-agent learning. The diversified agent-agent interaction data are generated by agents that have different ranks of policies. Concretely, we maintain a look-up memory during training, where each entry is a key and the corresponding value is a list. We take the training episode return of the agents' policies as the key, and use the list to store policies of the agents evaluated. When agents in the substrate start a new episode, there is a probability p to replace all agents' behavior policies with the sampled policies from the memory. This method is termed Ranked Policy Memory (RPM). Below we introduce how to build it and how to sample policies from it.

RPM Building. We denote an RPM with Ψ, which consists of $|R_{\text{max}}|$ entries, *i.e.*, ranks, where $|R_{\text{max}}|$ is the maximum training episode return (the episode return in the substrate). While agents are acting in the substrate, the training episode return R of all agents (with policies $\{\pi^i_\theta\}_{i=1}^N$) is returned. Then $\{\pi^i_\theta\}_{i=1}^N$ are saved into Ψ by appending agents' policies into the corresponding memory slot, $\Psi[r_e]\texttt{.add}(\{\pi^i_e\}_{i=1}^N)$. To avoid there being too many entries in the policy memory caused by continuous episode return values, we discretize the training episode return. The discretized entry κ covers a range of $[\kappa, \kappa+\psi)$, where $\psi > 0$ and is an integer number. For the training episode return R, the corresponding entry κ can be calculated by:

$$\kappa = \begin{cases} \lfloor R/\psi \rfloor \times \mathbf{1}\{(R \bmod \psi) \neq 0\} \times \psi, & \text{if } R \geq 0, \\ \lfloor R/\psi \rfloor \times \psi, & \text{otherwise.} \end{cases} \tag{2}$$

where $\mathbf{1}\{\cdot\}$ is the indicator function, and $\lfloor\cdot\rfloor$ is the floor function. Intuitively, discretizing R saves memory and memorize policies of similar performance in to the same rank. Therefore, diversified policies can be saved to be sampled for agents.

RPM Sampling. The memory Ψ stores diversified policies with different levels of performance. We can sample various policies of different ranks and assign each policy to each agent in the substrate to collect multi-agent trajectories for training. These diversified multi-agent trajectories can resemble trajectories generated by the interaction with agents possessing unknown policies in the evaluation scenario. At the beginning of an episode, we first randomly sample N keys with replacement and then randomly sample one policy for each key from the corresponding list. All agents' policies will be replaced with the newly sampled policies for multi-agent interactions in the substrate, thus generating diversified multi-agent trajectories.

MARL with RPM. We showcase an example of the workflow of RPM in Figure 3. There are three agents in training. Agents sample policies from RPM. Then all agents collect data in the substrate for training. The training episode return is then used to update RPM. During evaluation, agents

1 and 2 are selected as focal agents and agent 3 is selected as the background agent. We present the pseudo-code of MARL training with RPM in Algorithm 1. In Lines 4-5, the $\boldsymbol{\pi}_{\boldsymbol{\theta}_b}$ is updated by sampling policies from RPM. Then, new trajectories of $\mathcal{D}$ are collected in Line 6. $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ is trained in Line 7 with MARL method by using the newly collected trajecotries and $\boldsymbol{\pi}_{\boldsymbol{\theta}_b}$ is updated with the newly updated $\boldsymbol{\pi}_{\boldsymbol{\theta}}$. RPM is updated in Line 8. After that, the performance of $\boldsymbol{\pi}_{\boldsymbol{\theta}}$ is evaluated in the evaluation scenario $\mathcal{G}'$ and the evaluation score $\bar{R}$ is returned in Line 9.

Algorithm 1: MARL with RPM

1 **Input**: Initialize $\boldsymbol{\pi}_{\boldsymbol{\theta}}$, Ψ, $\mathcal{D}$, $\mathcal{G}$ and $\mathcal{G}'$;
2 **Input**: Initialize behavior policy $\boldsymbol{\pi}_{\boldsymbol{\theta}_b} \leftarrow \boldsymbol{\pi}_{\boldsymbol{\theta}}$;
3 **for** t in $1, \ldots,$ `MAX_STEP` **do**
4 **if** `RPM sampling` **then**
5   $\boldsymbol{\pi}_{\boldsymbol{\theta}_b} \leftarrow$ `SamplingRPM(`Ψ`)`;
6 $\mathcal{D} \leftarrow$ `GatherTrajectories(`$\boldsymbol{\pi}_{\boldsymbol{\theta}_b}, \mathcal{G}$`)`;
7 $\boldsymbol{\pi}_{\boldsymbol{\theta}} \leftarrow$ `MARLTrainig(`$\boldsymbol{\pi}_{\boldsymbol{\theta}}, \mathcal{D}$`)`;
8 $\Psi \leftarrow$ `UpdateRPM(`$\boldsymbol{\pi}_{\boldsymbol{\theta}}, \Psi, \mathcal{G}$`)`;
9 $\bar{R} \leftarrow$ `Evaluate(`$\boldsymbol{\pi}_{\boldsymbol{\theta}}, \mathcal{G}'$`)`;
10 $\boldsymbol{\pi}_{\boldsymbol{\theta}_b} \leftarrow \boldsymbol{\pi}_{\boldsymbol{\theta}}$;
11 **Output**: $\boldsymbol{\pi}_{\boldsymbol{\theta}}$.

Discussion. RPM leverages agents' previously trained models in substrates to cover as many patterns of multi-agent interactions as possible to achieve generalization of MARL agents when paired with agents with unseen policies in evaluation scenarios. It uses the self-play framework for data collection. Self-play (Brown, 1951; Heinrich et al., 2015; Silver et al., 2018; Baker et al., 2019) maintains a memory of the opponent's previous policies for acquiring equilibria. RPM differs from other self-play methods in four aspects: (i) self-play utilizes agent's previous policies to create fictitious opponents when the real opponents are not available. By playing with the fictitious opponents, many fictitious data are generated for training the agents. In RPM, agents load their previous policies to diversify the multi-agent interactions, such as multi-agent coordination and social dilemmas, and all agents' policies are trained by utilizing the diversified multi-agent data. (ii) Self-play does not maintain explicit ranks for policies while RPM maintains ranks of policies. (iii) Self-play was not introduced for generalization of MARL while RPM aims to improve the generalization of MARL. In Section 5, we also present the evaluation results of a self-play method.

4.2 MARL Training

We incorporate RPM into the MARL training pipeline. We take MAPPO (Yu et al., 2021) for instantiating our method, which is a multi-agent variant of PPO (Schulman et al., 2017) and outperforms many MARL methods (Rashid et al., 2018; 2020; Wang et al., 2021a) in various complex multi-agent domains. In MAPPO, a central critic is maintained for utilizing the concealed information of agents to boost multi-agent learning due to non-stationarity. RPM introduces a novel method for agents to collect experiences/trajectories $\boldsymbol{\tau} = \{\tau_i\}_{i=1}^N$. Each agent optimizes the following objective:

$$\mathcal{J}(\theta_i) = \mathbb{E}\left[\texttt{min}\left(\eta_i^t\left(\theta_i^t\right) \cdot A_i^t, \texttt{clip}\left(\eta_i^t\left(\theta_i^t\right), 1-\epsilon, 1+\epsilon\right) \cdot A_i^t\right)\right], \tag{3}$$

where $\eta_i^t(\theta_i^t) = \frac{\pi_{\theta_i^t}(u_i^t|\tau_i^t)}{\pi_{\theta_i^{\text{old}}}(u_i^t|\tau_i^t)}$ denotes the important sampling weight. The $\texttt{clip}\left(\cdot\right)$ clips the values of θ^i that are outside the range $[1-\epsilon, 1+\epsilon]$ and ϵ is a hyperparameter. A_i^t is a generalized advantage estimator (GAE) (Schulman et al., 2015). To optimize the central critic $V_\psi(\{o_i^t, u_i^t\}_{i=1}^N)$, we mix agents' observation-action pairs and output an N-head vector where each value corresponds to the agent's value:

$$\mathcal{L}(\psi) := \mathbb{E}_{\mathcal{D}' \sim \mathcal{D}}\left[\left(y_t - V_{\bar{\psi}}(\{o_i^t, u_i^t\}_{i=1}^N)\right)^2\right], \tag{4}$$

where $y_t = \left[\sum_{l=0}^{k-1} \gamma^l r_i^{t+l} + \gamma^k V_{\bar{\psi}}(\{o_i^{t+k}, u_i^{t+k}\}_{i=1}^N)\,[i]\right]_{i=1}^N$ is a vector of k-step returns, and $\mathcal{D}'$ is a sample from the replay buffer $\mathcal{D}$. In complex scenarios, *e.g.*, Melting Pot, with an agent's observation as input, its action would not impact other agents' return, since the global states contain redundant information that deteriorates multi-agent learning. We present the whole training process, the network architectures of the agent and the central critic in Appendix D.

5 Experiments

In this section, to verify the effectiveness of RPM in improving generalization of MARL, we conduct extensive experiments on Melting pot and present the empirical results. We first introduce Melting Pot, baselines and experiments setups. Then we present the main results to show the superiority

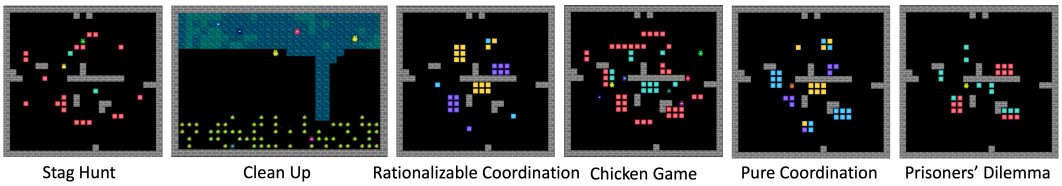


Figure 4: Melting Pot environments. More information can be found in Appendix A.

Table 1: Properties of Melting Pot environments (substrates and evaluation scenarios). The first column shows the properties and the first row lists environments. ✓ mark indicates the environment possessing the corresponding property while ✗ mark stands for the environment that does not own the corresponding property. Refer to Appendix A for more information about the environments.

	Stag Hunt	**Pure Coordination**	**Clean Up**	**Prisoners' Dilemma**	**Rational Coordination**	**Chicken Game**
Temporal Coordination	✗	✗	✓	✗	✗	✗
Reciprocity	✓	✓	✓	✓	✗	✓
Deception	✓	✗	✓	✓	✗	✓
Fair Resource Sharing	✗	✗	✓	✗	✗	✗
Convention Following	✓	✓	✓	✗	✓	✓
Task Partitioning	✗	✗	✓	✓	✗	✗
Trust & Partnership	✓	✗	✗	✗	✗	✓
Free Riding	✗	✗	✓	✗	✗	✗

of RPM. To demonstrate that ψ is important for RPM, we conducted ablation studies. We finally showcase a case study to visualize RPM. To sum up, we answer the following questions: **Q1**: Is RPM effective in boosting the generalization performance of MARL agents? **Q2**: Does the value of ψ matter in RPM for training? **Q3**: Does RPM gather diversified policies and trajectories?

5.1 Experimental Setup

Melting Pot. To demonstrate that RPM enables MARL agents to learn generalizable behaviors, we carry out extensive experiments on DeepMind's Melting Pot (Leibo et al., 2021). Melting Pot is a suite of testbeds for the generalization of MARL methods. It proposes a novel evaluation pipeline for the evaluation of the MARL method in various domains. That is, all MARL agents are trained in the substrate; during evaluation, some agents are selected as the focal agents (the agents to be evaluated) and the rest agents become the background agents (pretrained policies of MARL models will be plugged in); the evaluation scenarios share the same physical properties with the substrates. Melting Pot environments possess many properties, such as temporal coordination and free riding as depicted in Table 1. MARL agent performing well in these environments means its behaviors demonstrate these properties. In Figure 5, the agent's observation is shown in the green box to the lower left of the state (*i.e.*, the whole image). The agent is in the lower middle of the observation. The neural network architecture of the agent's policy is shown on the left. More information about the setting of substrates, neural network architectures, MARL training can be found in Appendix D.

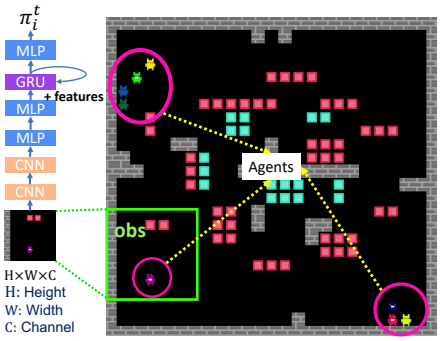


Figure 5: The green box to the lower left shows the agent's observation.

Baselines. Our baselines are MAPPO (Yu et al., 2021), MAA2C (Papoudakis et al., 2021), OPRE (Vezhnevets et al., 2020), heuristic fictitious self-play (HFSP) (Heinrich, 2017; Berner et al., 2019) and RandNet (Lee et al., 2019). MAPPO and MAA2C are MARL methods that achieved outstanding performance in various multi-agent scenarios (Papoudakis et al., 2021). OPRE was proposed for the generalization of MARL. RandNet is a general method for the generalization of RL by introducing a novel component in the convolutional neural network. HFSP is a general self-play method for obtaining equilibria in competitive games, we use it by using the policies saved by RPM.

Training setup. We use 6 representative substrates (Figure 4) to train MARL policies and choose one evaluation scenario from each substrate as our evaluation testbed. The properties of the environments

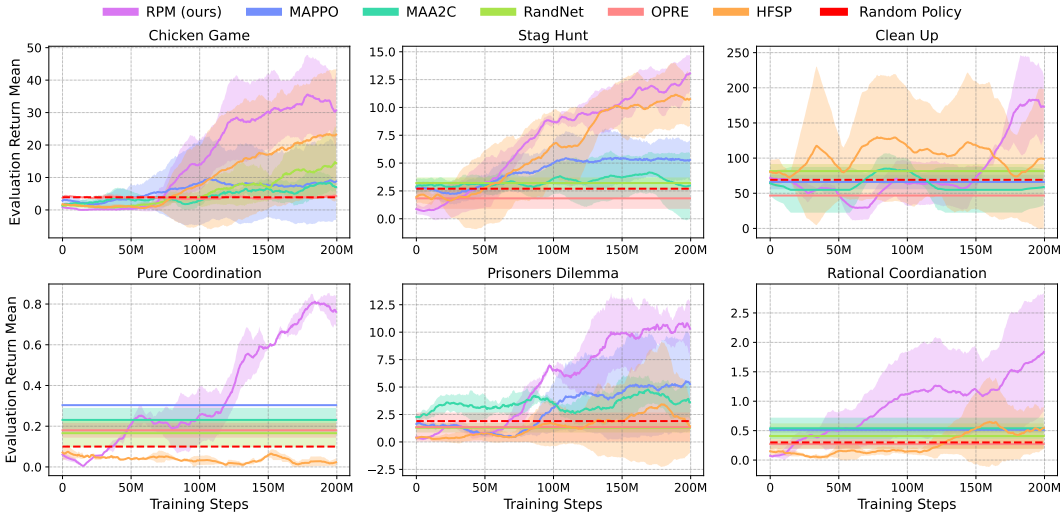


Figure 6: Evaluation results of RPM and baselines on evaluation scenarios. The red dash horizontal lines indicate the results of random policy.

are listed in Table 1. We train agents in Melting Pot substrates for 200 million frames with 3 random seeds for RPM and 4 seeds for baselines. Our training framework is a distributed framework where there are 30 CPU cores (actors) to collect experiences and 1 GPU for the learner to to learn policies. We implement our actors with Ray (Moritz et al., 2018) and the learner with EPyMARL (Papoudakis et al., 2021). We use mean-std to measure the performance of all methods. The bold lines in all figures are the mean and the shades stand for the standard deviation. Due to a limited computation budget, it is redundant in computation to compare our method with other methods such as QMIX (Rashid et al., 2018) and MADDPG (Lowe et al., 2017) as MAPPO outperforms them. All experiments are conducted on NVIDIA A100 GPUs.

5.2 Results

To answer **Q1**, we present the evaluation results of 6 Melting Pot evaluation scenarios in Figure 6. We can find that our method can boost MARL in various evaluation scenarios which have different properties as shown in Table 1. In Chicken Game (eval, 'eval' stands for the evaluation scenario of the substrate Chicken Game), RPM outperforms its counterparts with a convincing margin. HFSP attains over 20 evaluation mean returns. RandNet gets around 15 evaluation mean returns. MAA2C and OPRE perform nearly random (the red dash lines indicate the random result). In Pure Coordination (eval), Rational Coordination (eval) and Prisoners' Dilemma (eval), most of the baselines perform poorly. In Stag Hunt (eval) and Clean Up (eval), MAPPO and MAA2C also perform unsatisfactorily. We can also find that HFSP even gets competitive performance in Stag Hunt (eval) and Clean Up (eval). However, HFSP performs poorly in Pure Coordination (eval), Rational Coordination (eval) and Prisoners' Dilemma (eval). Therefore, vanilla self-play method cannot directly be applied to improve the generalization of MARL methods. To sum up, RPM boosts the performance up to around 402% on average compared with MAPPO on 6 evaluation scenarios.

5.3 Ablation Study

To investigate which value of ψ has the greatest impact on RPM performance, we conduct ablation studies by (i) removing ranks and sampling from the checkpoint directly; (ii) reducing the number of ranks by changing the value of ψ. As shown in Figure 7, without ranks (sampling policies without ranks randomly), RPM cannot perform well in all evaluation scenarios. Especially in Pure Coordination (eval) where the result is low and has large variance. In RPM, choosing the right interval ψ can im-

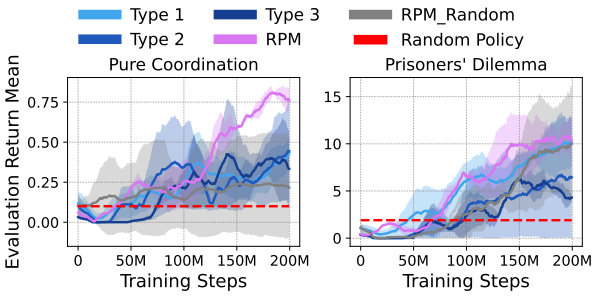
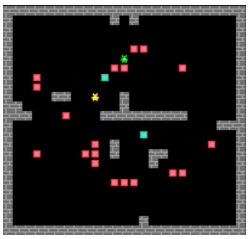
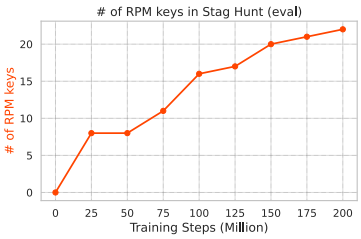


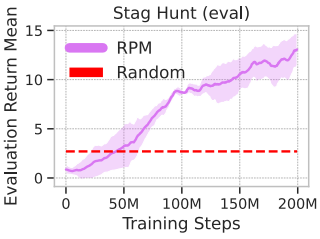
Figure 7: Ablation Studies: the performance of RPM with 3 types of ψ and Random sampling (without ranks).



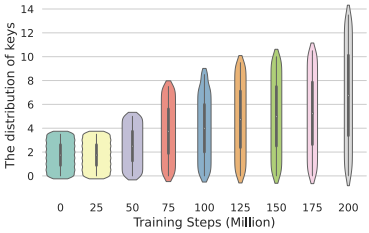
(a) Stag Hunt (eval)



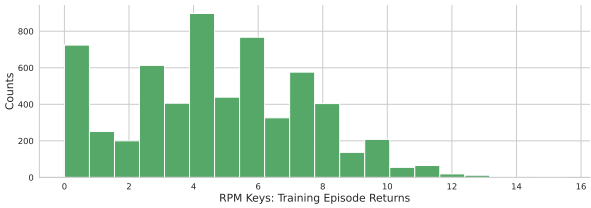
(b) The number of RPM keys



(c) RPM evaluation results



(d) The distribution of the keys of RPM



(e) The final histogram of the keys of RPM

Figure 8: Result analysis. **(a)** The pictorial view of Stag Hunt (eval); **(b)** The number of RPM keys during training; **(c)** The evaluation results of RPM on Stag Hunt (eval); **(d)** The distribution of the keys of RPM during training; **(e)** The histogram of the keys of RPM at timestep 200M during training.

prove the performance as shown in the results of Pure Coordination (eval) and Prisoners' Dilemma (eval), showing that the value of ψ is important for RPM. We summarize the results and values of ψ in Table 2 and Table 3.

Table 2: Ablation study: the final mean evaluation episode return. Curves are in Figure 7.

Eval Scenarios	RPM	Random	Types of ψ		
			1	2	3
Pure Coordination	0.76	0.22	0.40	0.42	0.36
Prisoners' Dilemma	10.56	9.78	10.0	6.35	4.24

Table 3: ψ values. ψ^* indicates the values of ψ used to get results in Figure 6.

Eval Scenarios	ψ^*	Types of ψ		
		1	2	3
Pure Coordination	0.01	0.1	0.5	1
Prisoners' Dilemma	0.02	0.2	1	5

5.4 Case Study

We showcase how RPM helps to train the focal agents to choose the right behaviors in the evaluation scenario after training in the substrate. To illustrate the trained performance of RPM agents, we use the RPM agent trained on Stag Hunt and run the evaluation on Stag Hunt (eval). In Stag Hunt, there are 8 agents in this environment. Each agent collects resources that represent 'hare' (red) or 'stag' (green) and compares inventories in an interaction, *i.e.*, encounter. The results of solving the encounter are the same as the classic Stag Hunt matrix game. In this environment, agents are facing tension between the reward for the team and the risk for the individual. In Stag Hunt (eval) (Figure 8 (a)). One focal agent interacts with seven pretrained agents. All background agents were trained to play the 'stag' strategy during the interaction[1]. The optimal policy for the focal agent is also to play 'stag'. However, it is challenging for agents to detect other agents' strategy since such a behavior may not persist in the substrate. Luckily, RPM enables focal agents to behave correctly in this scenario.

To answer **Q3**, we present the analysis of RPM on the substrate Stag Hunt and its evaluation scenario Stag Hunt (eval) in Figure 8. We can find that in Figure 8 (b), the number of the keys in RPM is growing monotonically during training and the maximum number of the keys in RPM is over 20, showing that agents trained with RPM discover many novel patterns of multi-agent interaction and new keys are created and the trained models are saved in RPM. Meanwhile, the evaluation performance is also increasing in Stag Hunt (eval) as depicted in Figure 8 (c). In Figure 8 (d), it is interesting to see that the distribution of the keys of RPM is expanding during training. In the last 25 million training steps, the last distribution of RPM keys covers all policies of different levels of performance, ranging from 0 to 14. By utilizing RPM, agents can collect diversified multi-agent trajectories for multi-agent training. Figure 8 (e) demonstrates the final histogram of RPM keys after training. There are over 600 trained policies that have small value of keys. Since at the early

[1]This preference was trained with pseudo rewards by Leibo et al. (2021) and the trained models are available at this link: `https://github.com/deepmind/meltingpot`

stage of training, agents should explore the environment, it is reasonable to find that a large number of trained policies of RPM keys have low training episode returns. After 50 million training steps, RPM has more policies that have higher training episode returns. Note that the maximum training episode return of RPM keys is over 14 while the maximum mean evaluation return of RPM shown in Figure 8 (c) is around 14.

In our experiments, we find that training policies with good performance in the substrate is crucial for improving generalization performance in the evaluation scenarios. When MARL agents perform poorly, *i.e.*, execute sub-optimal actions, the evaluation performance will be also inferior or even random, making it is hard to have diversified policies. We show the results in Appendix E.

6 Related Works

Recent advances in MARL (Yang & Wang, 2020; Zhang et al., 2021) have demonstrated its success in various complex multi-agent domains, including multi-agent coordination (Lowe et al., 2017; Rashid et al., 2018; Wang et al., 2021b), real-time strategy (RTS) games (Jaderberg et al., 2019; Berner et al., 2019; Vinyals et al., 2019), social dilemma (Leibo et al., 2017; Wang et al., 2018; Jaques et al., 2019; Vezhnevets et al., 2020), multi-agent communication (Foerster et al., 2016; Yuan et al., 2022), asynchronous multi-agent learning (Amato et al., 2019; Qiu et al., 2022), open-ended environment (Stooke et al., 2021), autonomous systems (Hüttenrauch et al., 2017; Peng et al., 2021) and game theory equilibrium solving (Lanctot et al., 2017; Perolat et al., 2022). Despite strides made in MARL, training generalizable behaviors in MARL is yet to be investigated.

Generalization in RL (Packer et al., 2018; Song et al., 2019; Ghosh et al., 2021; Lyle et al., 2022) has achieved much progress in domain adaptation (Higgins et al., 2017) and procedurally generated environments (Lee et al., 2019; Igl et al., 2020; Zha et al., 2020) in recent years. However, there are few works of generalization in MARL domains (Carion et al., 2019; Vezhnevets et al., 2020; Mahajan et al., 2022; McKee et al., 2022). Recently, Vezhnevets et al. (2020) propose a hierarchical MARL method for agents to play against opponents it hasn't seen during training. However, the evaluation scenarios are only limited to simple competitive scenarios. Mahajan et al. (2022) investigated the generalization in MARL empirically and proposed theoretical findings based on successor features (Dayan, 1993; Barreto et al., 2018). However, there is no method proposed to achieve generalization in MARL.

Ad-hoc team building (Stone & Kraus, 2010; Gu et al., 2021) models the multi-agent problem as a single-agent learning task. In ad-hoc team building, one ad-hoc agent is trained by interacting with agents that have fixed pretrained policies and the non-stationarity issue is not severe. However, in our formulation, non-stationarity is the main obstacle to MARL training. In addition, there is only one ad-hoc agent evaluated by interacting agents that are unseen during training while there can be more than one focal agent in our formulation as defined in Definition 2, thus making our formulation general and challenging. There has been a growing interest in applying self-play to solve complex games (Heinrich et al., 2015; Silver et al., 2018; Hernandez et al., 2019; Baker et al., 2019); however, its value in enhancing the generalization of MARL agents has yet to be examined.

7 Conclusion, Limitations and Future Work

In this paper, we consider the problem of achieving generalizable behaviors in MARL. We first model social learning with Markov Game. In order to train agents that are able to interact with agents that possess unseen policies. We propose a simple yet effective method, RPM, to save policies of different levels. We save policies by ranking the training episode return. Empirically, RPM significantly boosts the performance of MARL agents in a variety of Melting Pot evaluation scenarios.

RPM's performance is highly dependent on the appropriate value of ψ. Several attempts are required to determine the correct value of ψ for RPM. We are interested in discovering more broad measures for ranking policies that do not explicitly consider the training episode return. Recently, there is a growing interest in planning in RL, especially with model-based RL. We are interested in exploring the direction of applying planning and opponent/teammate modelling for attaining generalized behaviors with MARL for future work. In multi-agent scenarios, agents are engaged in complex interactions. Devising novel self-play method is our future direction for improving generalization of MARL methods.

8 ETHICS STATEMENT

We addressed the relevant aspects in our conclusion and have no conflicts of interest to declare.

9 REPRODUCIBILITY STATEMENT

We provide detailed descriptions of our experiments in the appendix and list all relevant parameters in Table 4 and Table 5 in Appendix D. The code can be found at this anonymous link: `https://sites.google.com/view/rpm-iclr2023/`.

REFERENCES

Christopher Amato, George Konidaris, Leslie P Kaelbling, and Jonathan P How. Modeling and planning with macro-actions in decentralized pomdps. *Journal of Artificial Intelligence Research*, 64:817–859, 2019.

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 31, 2017.

Bowen Baker, Ingmar Kanitscheider, Todor Markov, Yi Wu, Glenn Powell, Bob McGrew, and Igor Mordatch. Emergent tool use from multi-agent autocurricula. In *International Conference on Learning Representations*, 2019.

Andre Barreto, Diana Borsa, John Quan, Tom Schaul, David Silver, Matteo Hessel, Daniel Mankowitz, Augustin Zidek, and Remi Munos. Transfer in deep reinforcement learning using successor features and generalised policy improvement. In *International Conference on Machine Learning*, pp. 501–510. PMLR, 2018.

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

George W Brown. Iterative solution of games by fictitious play. *Act. Anal. Prod Allocation*, 13(1): 374, 1951.

Nicolas Carion, Nicolas Usunier, Gabriel Synnaeve, and Alessandro Lazaric. A structured prediction approach for generalization in cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 32, 2019.

Kyunghyun Cho, Bart Van Merriënboer, Dzmitry Bahdanau, and Yoshua Bengio. On the properties of neural machine translation: Encoder-decoder approaches. *arXiv preprint arXiv:1409.1259*, 2014.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

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

Jakob Foerster, Ioannis Alexandros Assael, Nando de Freitas, and Shimon Whiteson. Learning to communicate with deep multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems*, pp. 2137–2145, 2016.

Scott Fujimoto, Herke Hoof, and David Meger. Addressing function approximation error in actor-critic methods. In *International conference on machine learning*, pp. 1587–1596. PMLR, 2018.

Dibya Ghosh, Jad Rahme, Aviral Kumar, Amy Zhang, Ryan P Adams, and Sergey Levine. Why generalization in rl is difficult: Epistemic pomdps and implicit partial observability. *Advances in Neural Information Processing Systems*, 34, 2021.

Pengjie Gu, Mengchen Zhao, Jianye Hao, and Bo An. Online ad hoc teamwork under partial observability. In *International Conference on Learning Representations*, 2021.

Johannes Heinrich. *Reinforcement learning from self-play in imperfect-information games*. PhD thesis, UCL (University College London), 2017.

Johannes Heinrich, Marc Lanctot, and David Silver. Fictitious self-play in extensive-form games. In *International conference on machine learning*, pp. 805–813. PMLR, 2015.

Daniel Hernandez, Kevin Denamganaï, Yuan Gao, Peter York, Sam Devlin, Spyridon Samothrakis, and James Alfred Walker. A generalized framework for self-play training. In *2019 IEEE Conference on Games (CoG)*, pp. 1–8. IEEE, 2019.

Irina Higgins, Arka Pal, Andrei Rusu, Loic Matthey, Christopher Burgess, Alexander Pritzel, Matthew Botvinick, Charles Blundell, and Alexander Lerchner. Darla: Improving zero-shot transfer in reinforcement learning. In *International Conference on Machine Learning*, pp. 1480–1490. PMLR, 2017.

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Dieuwke Hupkes, Verna Dankers, Mathijs Mul, and Elia Bruni. Compositionality decomposed: how do neural networks generalise? *Journal of Artificial Intelligence Research*, 67:757–795, 2020.

Maximilian Hüttenrauch, Adrian Šošić, and Gerhard Neumann. Guided deep reinforcement learning for swarm systems. *In AAMAS 2017 Autonomous Robots and Multirobot Systems (ARMS) Workshop*, 2017.

Maximilian Igl, Gregory Farquhar, Jelena Luketina, Wendelin Boehmer, and Shimon Whiteson. Transient non-stationarity and generalisation in deep reinforcement learning. In *International Conference on Learning Representations*, 2020.

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.

Natasha Jaques, Angeliki Lazaridou, Edward Hughes, Caglar Gulcehre, Pedro Ortega, DJ Strouse, Joel Z Leibo, and Nando De Freitas. Social influence as intrinsic motivation for multi-agent deep reinforcement learning. In *International Conference on Machine Learning*, pp. 3040–3049. PMLR, 2019.

Robert Kirk, Amy Zhang, Edward Grefenstette, and Tim Rocktäschel. A survey of generalisation in deep reinforcement learning. *arXiv preprint arXiv:2111.09794*, 2021.

Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in Neural Information Processing Systems*, 30, 2017.

Kimin Lee, Kibok Lee, Jinwoo Shin, and Honglak Lee. Network randomization: A simple technique for generalization in deep reinforcement learning. In *International Conference on Learning Representations*, 2019.

Joel Z Leibo, Vinicius Zambaldi, Marc Lanctot, Janusz Marecki, and Thore Graepel. Multi-agent reinforcement learning in sequential social dilemmas. *arXiv preprint arXiv:1702.03037*, 2017.

Joel Z Leibo, Edgar A Dueñez-Guzman, Alexander Vezhnevets, John P Agapiou, Peter Sunehag, Raphael Koster, Jayd Matyas, Charlie Beattie, Igor Mordatch, and Thore Graepel. Scalable evaluation of multi-agent reinforcement learning with melting pot. In *International Conference on Machine Learning*, pp. 6187–6199. PMLR, 2021.

Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *International Conference on Learning Representations*, 2016.

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

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems*, pp. 6379–6390, 2017.

Clare Lyle, Mark Rowland, Will Dabney, Marta Kwiatkowska, and Yarin Gal. Learning dynamics and generalization in reinforcement learning. *arXiv preprint arXiv:2206.02126*, 2022.

Anuj Mahajan, Mikayel Samvelyan, Tarun Gupta, Benjamin Ellis, Mingfei Sun, Tim Rocktäschel, and Shimon Whiteson. Generalization in cooperative multi-agent systems. *arXiv preprint arXiv:2202.00104*, 2022.

Kevin R McKee, Joel Z Leibo, Charlie Beattie, and Richard Everett. Quantifying the effects of environment and population diversity in multi-agent reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 36(1):1–16, 2022.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

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.

Philipp Moritz, Robert Nishihara, Stephanie Wang, Alexey Tumanov, Richard Liaw, Eric Liang, Melih Elibol, Zongheng Yang, William Paul, Michael I Jordan, et al. Ray: A distributed framework for emerging {AI} applications. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, pp. 561–577, 2018.

Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

Charles Packer, Katelyn Gao, Jernej Kos, Philipp Krähenbühl, Vladlen Koltun, and Dawn Song. Assessing generalization in deep reinforcement learning. *arXiv preprint arXiv:1810.12282*, 2018.

Georgios Papoudakis, Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Benchmarking multi-agent deep reinforcement learning algorithms in cooperative tasks. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 1)*, 2021.

Zhenghao Peng, Quanyi Li, Ka Ming Hui, Chunxiao Liu, and Bolei Zhou. Learning to simulate self-driven particles system with coordinated policy optimization. *Advances in Neural Information Processing Systems*, 34:10784–10797, 2021.

Julien Perolat, Bart de Vylder, Daniel Hennes, Eugene Tarassov, Florian Strub, Vincent de Boer, Paul Muller, Jerome T Connor, Neil Burch, Thomas Anthony, et al. Mastering the game of stratego with model-free multiagent reinforcement learning. *arXiv preprint arXiv:2206.15378*, 2022.

Wei Qiu, Weixun Wang, Rundong Wang, Bo An, Yujing Hu, Svetlana Obraztsova, Zinovi Rabinovich, Jianye Hao, Yingfeng Chen, and Changjie Fan. Off-beat multi-agent reinforcement learning. *arXiv preprint arXiv:2205.13718*, 2022.

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, 2018.

Tabish Rashid, Gregory Farquhar, Bei Peng, and Shimon Whiteson. Weighted qmix: Expanding monotonic value function factorisation for deep multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 33:10199–10210, 2020.

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

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

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419): 1140–1144, 2018.

Xingyou Song, Yiding Jiang, Stephen Tu, Yilun Du, and Behnam Neyshabur. Observational overfitting in reinforcement learning. In *International Conference on Learning Representations*, 2019.

Peter Stone and Sarit Kraus. To teach or not to teach?: decision making under uncertainty in ad hoc teams. In *AAMAS*, pp. 117–124, 2010.

Adam Stooke, Anuj Mahajan, Catarina Barros, Charlie Deck, Jakob Bauer, Jakub Sygnowski, Maja Trebacz, Max Jaderberg, Michael Mathieu, et al. Open-ended learning leads to generally capable agents. *arXiv preprint arXiv:2107.12808*, 2021.

Robert Sugden. Rights, co-operation and welfare. In *The Economics of Rights, Co-operation and Welfare*, pp. 170–182. Springer, 2005.

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

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 112(1-2):181–211, 1999.

Richard Stuart Sutton. *Temporal credit assignment in reinforcement learning*. PhD thesis, University of Massachusetts Amherst, 1984.

Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente. Multiagent cooperation and competition with deep reinforcement learning. *PLoS ONE*, 12(4), 2017.

Alexander Vezhnevets, Yuhuai Wu, Maria Eckstein, Rémi Leblond, and Joel Z Leibo. Options as responses: Grounding behavioural hierarchies in multi-agent reinforcement learning. In *International Conference on Machine Learning*, pp. 9733–9742. PMLR, 2020.

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.

Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. QPLEX: Duplex dueling multi-agent q-learning. In *International Conference on Learning Representations*, 2021a.

Weixun Wang, Jianye Hao, Yixi Wang, and Matthew Taylor. Towards cooperation in sequential prisoner's dilemmas: a deep multiagent reinforcement learning approach. *arXiv preprint arXiv:1803.00162*, 2018.

Yihan Wang, Beining Han, Tonghan Wang, Heng Dong, and Chongjie Zhang. DOP: Off-policy multi-agent decomposed policy gradients. In *International Conference on Learning Representations*, 2021b.

Christopher JCH Watkins and Peter Dayan. Q-Learning. *Machine Learning*, 8(3-4):279–292, 1992.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Yaodong Yang and Jun Wang. An overview of multi-agent reinforcement learning from game theoretical perspective. *arXiv preprint arXiv:2011.00583*, 2020.

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.

Lei Yuan, Jianhao Wang, Fuxiang Zhang, Chenghe Wang, ZongZhang Zhang, Yang Yu, and Chongjie Zhang. Multi-agent incentive communication via decentralized teammate modeling. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(9):9466–9474, Jun. 2022.

Daochen Zha, Wenye Ma, Lei Yuan, Xia Hu, and Ji Liu. Rank the episodes: A simple approach for exploration in procedurally-generated environments. In *International Conference on Learning Representations*, 2020.

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

A ENVIRONMENTS

This section introduces Melting Pot in-depth, including substrates and evaluation scenarios.

A.1 GENERAL SETTINGS

Melting Pot (Leibo et al., 2021) is a suite of testbeds for MARL evaluation. It proposes a novel evaluation pipeline for evaluating the MARL method in various domains. That is, all MARL agents are trained in the substrate; during evaluation, some agents are selected as the focal agents (the agents to be evaluated), and the rest agents become the background agents (pretrained policies of MARL models will be plugged in); the evaluation scenarios share the same physical properties with the substrates. Melting Pot environments possess many properties, such as temporal coordination and free riding. MARL agent performing well in these environments means its behaviors demonstrate these properties. In each substrate, episodes last 1000 or 2000 steps. The agents have a partial observability window of 11×11 sprites. The agent can observe 9 rows in front of itself, 1 row behind, and 5 columns to either side. Sprites are 8×8 pixels. Thus, in RGB pixels, the size of each observation is $88 \times 88 \times 3$. All agents use RGB pixel representations as their inputs. In Figure 9, the agent's observation is shown in the green box to the lower left of the state (*i.e.*, the whole image). The agent is in the lower middle of the observation. The neural network architecture of the agent's policy is shown on the left. We introduce the neural network architecture design in Appendix C, MARL training and hyperparameters in Appendix D.

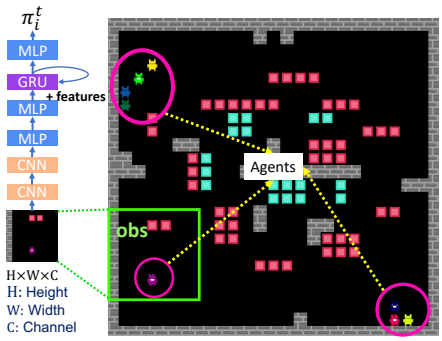


Figure 9: The green box to the lower left shows the agent's observation.

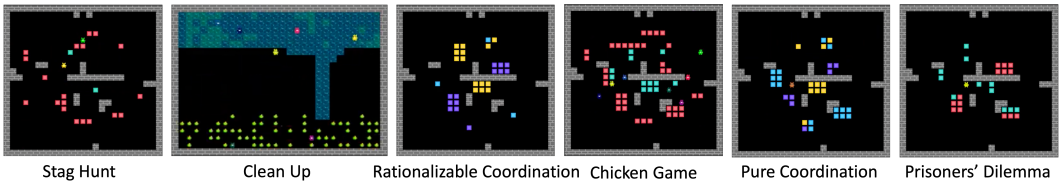


Figure 10: Melting Pot environments.

A.2 SUBSTRATES AND EVALUATION SCENARIOS

We introduce substrates and evaluation scenarios used in the experiments. In all substrates and scenarios, agents' movement actions are: `forward`, `backward`, `strafe left`, `strafe right`, `turn left`, `turn right`. Unless otherwise stated, each episode lasts 1000 steps. We show the environments in Figure 10, for readers' convenience.

Chicken Game. In this environment, there are 8 agents in the substrate. Agents move around the environments[2] and collect resources of 2 different colors. Each agent carries an inventory $\rho = (\rho_1, \rho_2)$ with the count of resources picked up since the last respawn. Due to partial observability, agents can only observe their inventory[3]. The more resources of a given type an agent picks up, the more committed the agent becomes to the pure strategy corresponding to that resource[4]. The agent can zap the other agent via its zapping beam for interaction. When an interaction occurs, a traditional matrix game is started. Here, in this environment, it is a Chicken Game (Sugden, 2005) where both agents

[2]There are two categories of environments: substrates and evaluation scenarios.

[3]It also applies to other environments where agents have inventories

[4]It also applies for other environments where matrix games should be resolved when two-agent interactions occur.

trying to exploit the other leads to the worst payoff, *i.e.*rewards, for both. Gathering red resources makes the agent's strategy towards committing 'hawk' while collecting green resources pushes it toward playing 'dove'. The payoff matrix for row and column players is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 3 & 2 \\ 5 & 0 \end{bmatrix}.$$

Chicken Game (eval). The task and the payoff matrix in this scenario are the same as in Chicken Game. In this scenario, one focal agent is joining seven background agents. Unlike the focal agent that can play any strategy, the background agents were pretrained with pseudo rewards to play 'dove'. The best strategy for the focal agent is to play 'hawk'.

Stag Hunt. Similar to Chicken Game, there are 8 agents in this environment. Each agent collects resources that represent 'hare' (red) or 'stag' (green) and compares inventories in an interaction, *i.e.*, encounter. The results of solving the encounter are the same as the classic Stag Hunt matrix game. In this environment, agents are facing tension between the reward for the team and the risk for the individual. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 4 & 0 \\ 2 & 2 \end{bmatrix}.$$

Stag Hunt (eval). In this environment, one agent interacts with seven pretrained agents. All background agents were trained to play the 'stag' strategy during the interaction. The optimal policy for the focal agent is also to play 'stag'.

Clean Up. There are seven agents in the environment. Agents are rewarded (+1) for collecting apples. In the environment, there are an orchard and a river. Agents should clean the river frequently to reduce pollution for the irrigation of the orchard. Apples in the orchard grow at a rate inversely related to the river's cleanliness. When the cleanliness rate reaches a certain threshold, apples stop growing. Agents can take `clean` action to clean a small amount of pollution from the river. However, such action only works in a small region around the agent in the river. So, agents should move to clean the river without any rewards. Consequently, agents should maintain the public good of orchard regrowth by cleaning the river. This creates a tension between the short-term individual incentive to maximize agents' reward by staying in the orchard and the long-term group interest in a clean river.

Clean Up (eval). In this evaluation scenario, three focal agents join four background agents. All background agents have been trained to behave altruistically, *i.e.*, always cleaning the river without consuming apples. Thus, the optimal policy for the focal agent is to collect as many apples as possible without moving out of the orchard to clean the river.

Pure Coordination. In this environment, eight agents cannot be identified as individuals because all agents look the same. Agents gather resources of three different colors. So, the size of the agent's inventory is 3. To maximize the reward, all agents should collect the same colored resource when the encounter occurs. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Pure Coordination (eval). In this evaluation scenario, there are seven focal agents and one background agent. The background agent has been trained to target one particular resource out of three colors of resources. Focal agents should observe other agents to see the resources other agents are collecting and then decide the right color to pick. This scenario aims to evaluate that agents' coordination is not disrupted by the presence of unfamiliar other agents who has a special preference for one particular colored resource.

Prisoners' Dilemma. Eight agents collect colored resources that represent 'defect' (red) or 'cooperate' (green). Agents compare their inventories in an encounter where a classic Prisoner's Dilemma matrix game is resolved. Agents face tension between the reward for the group and the reward for the individual. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 3 & 0 \\ 4 & 1 \end{bmatrix}.$$

Prisoners' Dilemma (eval). In this evaluation scenario, one focal agent joins seven background agents. All background agents will play cooperative strategies, *i.e.*, collecting 'cooperate' resources and rarely collecting 'defect'). The optimal policy for the focal agent is to identify such a pattern and then collect 'defect' resources.

Rational Coordination. The environment setting is the same as Pure Coordination, except that different colored resources are of different values. Agents should find the optimal color to maximize the group reward. The matrix for the interaction is:

$$\Phi_{\text{row}} = \Phi_{\text{col}}^{\text{T}} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 2 & 3 \end{bmatrix}.$$

Rational Coordination (eval). In this evaluation scenario, there are seven focal agents and one background agent. The background agent has been trained to target one particular resource out of three colors of resources. This scenario is similar to **Pure Coordination (eval)** since it aims to evaluate that agents' coordination is not disrupted by the presence of unfamiliar other agents who has a special preference for one particular colored resource. However, this scenario is more challenging than **Pure Coordination (eval)**. While focal agents' choices are better than miscoordination, some choices are better than coordinating for the focal agents.

B BASELINES

We introduce baselines trained and evaluated in the experiment in detail. Baselines are MAPPO (Yu et al., 2021), MAA2C (Papoudakis et al., 2021), OPRE (Vezhnevets et al., 2020), RandNet (Lee et al., 2019) and HFSP (Heinrich et al., 2015; Baker et al., 2019).

B.1 MAPPO

MAAPO is an extension of PPO (Schulman et al., 2017) for multi-agent RL. Following the CTDE (Oliehoek et al., 2008) training and execution paradigm, agents take actions independently during execution and agents' policies are trained via sharing information (*e.g.*,) with other agents. In MAPPO, there are N policies $\{\pi_i\}_{i=1}^N$ for each agent i. A central critic is maintained by feeding all agents' observations and actions $\{o_i^t, u_i^t\}_{i=1}^N$. Although the global state contains all agents' observations, it contains redundant information that deteriorates the central critic learning with TD-learning (Sutton, 1984). Note that all baselines that have a central critic takes all agents' observations and actions $\{o_i^t, u_i^t\}_{i=1}^N$ as the input.

B.2 MAA2C

MAA2C is a multi-agent RL variant of A2C (Mnih et al., 2016). MAA2C adopts the same training and execution paradigm used in MAPPO. Similar to A2C, TD error is used as the advantage in MAA2C for training agents' policies via maximizing policy gradient loss.

B.3 OPRE

We build OPRE (Vezhnevets et al., 2020) on top of MAPPO. The key idea behind OPRE is to re-use the same latent space to factorise the policy via creating a hierarchical policy structure:

$$\pi_i(u_i|o_i^{\leq t}, o') = \sum_z q(z|o')\eta(u_i|o_i^{\leq t}, z)$$

where o' is $\{o_i^t, u_i^t\}_{i=1}^N$ and $\eta(u_i|o_i^{\leq t}, z)$ is a mixture component of the policy, *i.e.*, an option. $o_i^{\leq t}$ can be represented via recurrent neural networks (Hochreiter & Schmidhuber, 1997; Cho et al., 2014). Note that the 'option' here differs from the *option* in hierarchical RL (Sutton et al., 1999; Bacon et al., 2017). In OPRE, the option has no explicit probability distribution of entering an option and no explicit probability distribution of exiting the current option. The behavior policy is defined as:

$$\mu_i(o_i^{\leq t}) = \sum_z p(z|o')\eta(u_i|o_i^{\leq t}, z)$$

Then $p(z|o')$ can be trained via $\texttt{KL}(q||p)$ together with the policy and the central critic in an end-to-end manner. We use the default hyperparameters used in OPRE in our experiments. The number of options is 16.

B.4 RANDNET

Lee et al. (2019) proposed RandNet for improving the generalization of RL in unseen environments, especially environments with new textures and layouts. RandNet utilizes a single-layer convolutional neural network (CNN) as a random network, where its output has the same dimension with the input. To reinitialize the parameters of the random network, RandNet utilizes the following mixture of distributions: $P(\phi) = \alpha\mathbb{I}(\phi = \mathbf{I}) + (1-\alpha)\mathcal{N}\left(\mathbf{0}; \sqrt{\frac{2}{n_{\text{in}}+n_{\text{out}}}}\right)$ where $\mathbf{I}$ is an identity kernel, $\alpha \in [0, 1]$ is a positive constant, $\mathcal{N}$ stands for the normal distribution. n_{in} and n_{out} are the number of input and output channels, respectively. We use RandNet in the policy network and the critic network of MAPPO. We use the default hyperparameters used in RandNet in our experiments.

B.5 HFSF

Self-play (Brown, 1951; Heinrich et al., 2015; Silver et al., 2018; Baker et al., 2019) has been studied for obtaining equilibria via creating fictitious plays by sampling agents' past policies. HFSP is a

heuristic fictitious self-play method. HFSP uses the MARL framework of MAPPO. Like RPM, it maintains a memory to save all the policies after each training step. HFSP agents have a probability of 0.7 to sample the lasted policies and a probability of 0.3 to sample previous policies. RPM can be considered as a ranked self-play by sampling policies with a hierarchy.

C ARCHITECTURES

We first introduce the neural network architecture of the policy, the critic and the training pipeline for all methods, and the hyperparameters used in the neural network architectures. RPM and all baselines use the same network architecture.

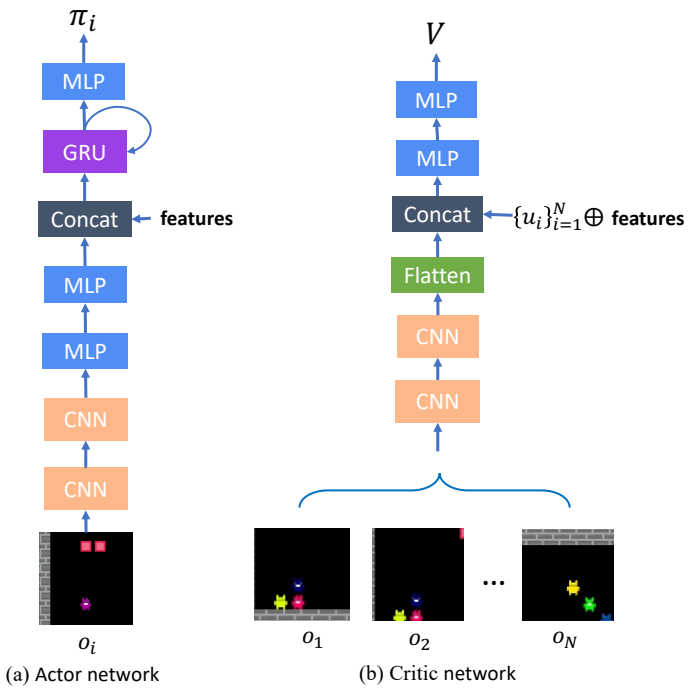


Figure 11: The networks of the policy (left) and the critic (right).

Actor Network. The actor network consists of a convolutional neural network (CNN) with two layers. The two CNN layers use the ReLU activation function. The first and the second layer have 16 and 32 output channels, 8 and 4 kernel shapes and 8 and 1 strides, respectively. An MLP follows the two CNN layers with two layers with 64 neurons each. The MLP uses the ReLU action function. It is then followed by a GRU (Cho et al., 2014) with 128 units. The input of the GRU is the concatenation of the output of the MLP and the features (such as the agent's position, orientation and inventory). The output of the GRU is fed into the MLP, and it outputs the policy π_i for agent i.

Critic Network. The critic network is shared by all agents. The critic network consists of a CNN with two layers. The two CNN layers use the ReLU activation function. The first and the second layer have 16 and 32 output channels, 8 and 4 kernel shapes and 8 and 1 strides, respectively. The CNN is then followed by a concatenation of all agents' actions and features (such as agent's position, orientation and inventory). The concatenation is then fed into an MLP with two layers with 64 neurons. The MLP uses the ReLU action function. The MLP outputs the value, a vector with the dimension of N. We take all agents' observations as a batch and feed them into the CNN. We then flatten the CNN's output and feed it with agents' actions and features as inputs to the MLP network to get the value vector for all agents.

Training. Our training framework is a distributed framework with 30 CPU cores to collect experiences and 1 GPU for the learner to learn policies, similar to the framework used in IMPALA (Espeholt et al., 2018). To improve the efficiency and save memory, we use parameter sharing (Rashid et al., 2018; Wang et al., 2021a; Yu et al., 2021), *i.e.*, all agents share a policy network. We adopt the CTDE framework to train the policies and the critic.

D TRAINING SETTINGS

We implement our method with Python and PyTorch. The learner is implemented with EPyMARL (Papoudakis et al., 2021) and the actors that collect experiments are implemented with Ray (Moritz et al., 2018). We train agents in Melting Pot substrates for 200 million frames with 3 random seeds for RPM and 4 seeds for baselines. We randomly sample policies from RPM. The discount factor $\gamma = 0.99$ and we follow the default hyper-parameters used in the original papers of all methods in our research. We carry out experiments on NVIDIA A100 Tensor Core GPU. We resort to mean-std values as our performance evaluation measurement. We use Adam as our optimizer. We list some important hyper-parameters in Table. 4.

Table 4: Hyper-parameters

hyper-parameter	**Value**
Optimizer	Adam
Learning rate	1e-4
Adam betas	$(0.9, 0.999)$
Adam epsilon	1e-8
Adam weight decay	0
Gradient norm clip	10
Batch size	60
Replay buffer size	600
γ	0.99
Evaluation interval	1,000
Target update interval	200
p	0.5

Table 5: The value of ψ

Melting Pot Substrate	**The value of ψ**
Stag Hunt	1
Pure Coordination	0.01
Clean Up	1
Prisoners' Dilemma	0.02
Rational Coordination	0.2
Chicken Game	1

E RESULTS

We depict the episode return within the substrate. During training, the MARL methods are evaluated in the substrate. Figure 12 demonstrates that despite the environments being distinct, RPM also demonstrates leading performance. Once the agents in the substrate achieve a satisfactory episode return, the trained policy will be saved at the appropriate rank. In turn, it improves the performance of RPM in the evaluation scenario by collecting diverse data on multi-agent interactions.

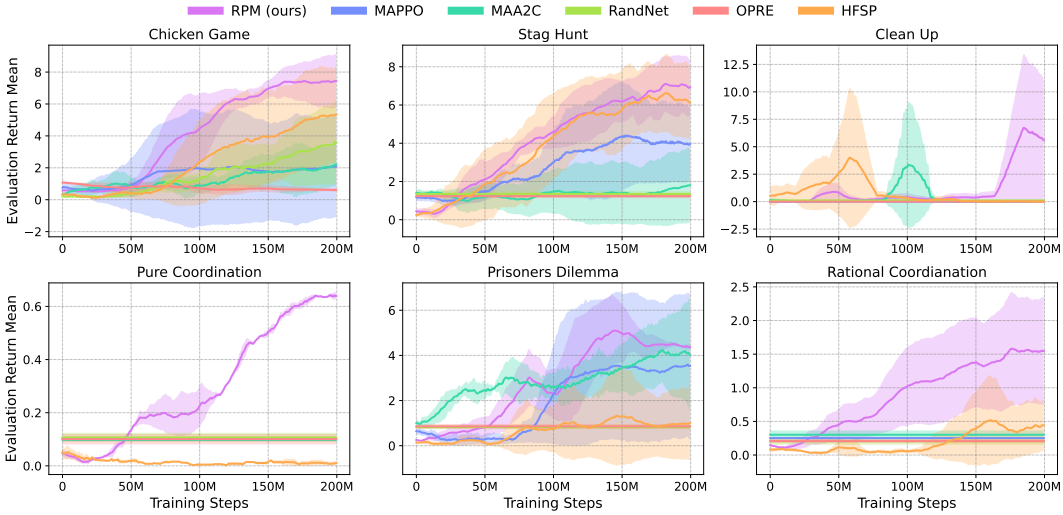


Figure 12: Episode return in substrates.