A ENVIRONMENTS

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

A.1 GENERAL SETTINGS

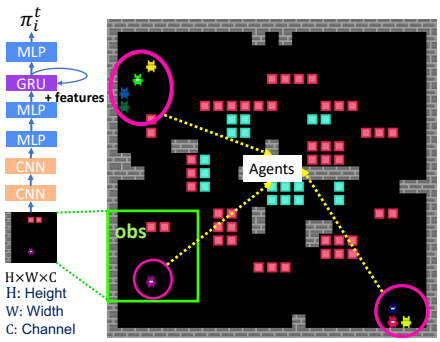

Figure 11: The green box to the lower left shows the agent's observation.

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 11, 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 Appx C, MARL training and hyperparameters in Appx D.

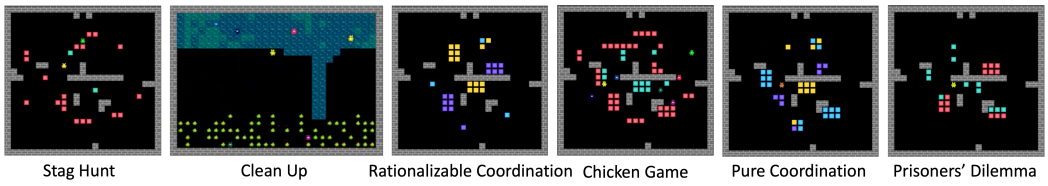

Figure 12: Melting Pot environments.

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 Fig. 12, 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

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

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 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 (CG) 1. 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'.

Chicken Game (CG) 2. The task and the payoff matrix in this scenario are the same as in Chicken Game. In this scenario, one focal agent joins seven background agents. The background agents were trained with alongside pure hawk and dove agents but were not given any non-standard pseudo rewards. They learned to play hawk to defect other agents who are collecting dove resources during the interaction.

Chicken Game (CG) 3. The task and the payoff matrix in this scenario are the same as in Chicken Game. In this scenario, two focal agents join five background agents. The background agents play dove unless and until they are defected on by a partner (*i.e.*, their partner chooses hawk). Subsequently, they will play hawk in each encounter for the remainder of the episode.

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 (SH) 1. 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'.

Stag Hunt (SH) 2. In this environment, one agent interacts with seven pretrained agents. All background agents were trained to play the 'hare' strategy during the interaction. The optimal policy for the focal agent is also to play 'hare'.

Stag Hunt (SH) 3. In this environment, two agent interacts with six pretrained agents. All background agents were trained to be reciprocators. They play stag. They are triggered to paly hare when their interaction partners play hare for the remainder of the episode.

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 (CU) 1. 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

[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.

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.

Clean Up (CU) 2. In this evaluation scenario, three focal agents join four background agents. All background agents start out cleaning in the first 250 steps. They alternate cleaning with eating every 250 steps. Focal agents should learn to take turns with the background agents.

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 (PC) 1. 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.

Pure Coordination (PC) 2. In this evaluation scenario, there is one focal agents and seven background agent. The background agents targets resource B.

Pure Coordination (PC) 3. In this evaluation scenario, there is one focal agents and seven background agent. The background agents targets resource C.

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 (PD) 1. 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.

Prisoners Dilemma (PD) 2. In this evaluation scenario, one focal agent joins seven background agents. The background agents are conditional cooperators. They collects 'cooperate' resources and cooperate with interaction partners. They stop cooperative strategies when they have been defected by their partners twice. After that, they collect 'defect' resources and strike back for the remainder of the episode.

Prisoners Dilemma (PD) 3. In this evaluation scenario, one focal agent joins seven background agents. The background agents are conditional cooperators. They collects 'cooperate' resources and cooperate with interaction partners. They stop cooperative strategies when they have been defected by their partners twice. After that, they collect 'defect' resources and strike back for the remainder of the episode. The optimal strategy for the focal agents is to cooperate when the episode nearly ends and then defect only once as there is no time for the background agents to revenge.

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 (RC) 1. 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 1** 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 1**. While focal agents' choices are better than miscoordination, some choices are better than coordinating for the focal agents.

Rational Coordination (RC) 2. In this evaluation scenario, there are seven focal agents and one background agent. The background agent has been trained to target the resource A. Because both resource B and C are better than resource A for every agent, it is not rational for all agents to coordinate on resource A.

Rational Coordination (RC) 3. In this evaluation scenario, there are four focal agents and four background agents. The background agent has been trained to target different resources, making them uncoordinated. In this case, it is rational that the focal agents should learn to observe partners' preferences and interact with familiar individuals who collect resources C.

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 HFSP

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.

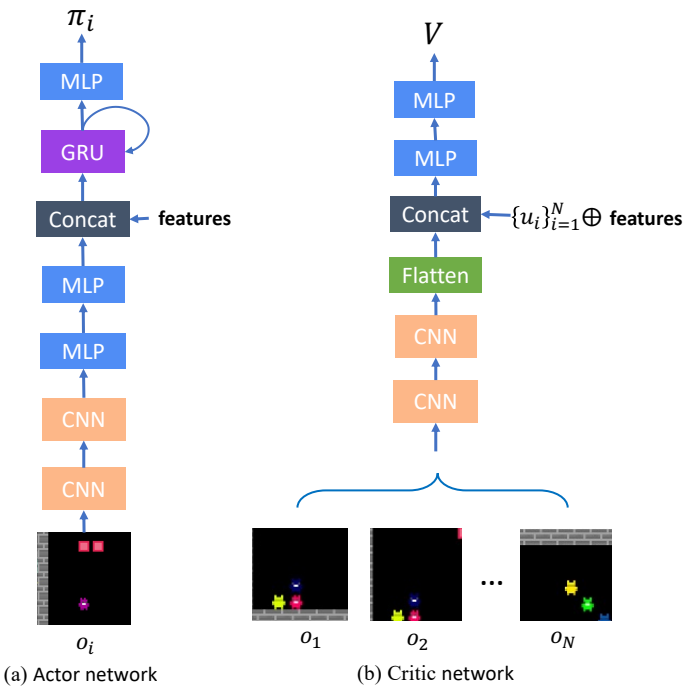

Figure 13: The networks of the policy (left) and the critic (right).

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. Fig. 14 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.

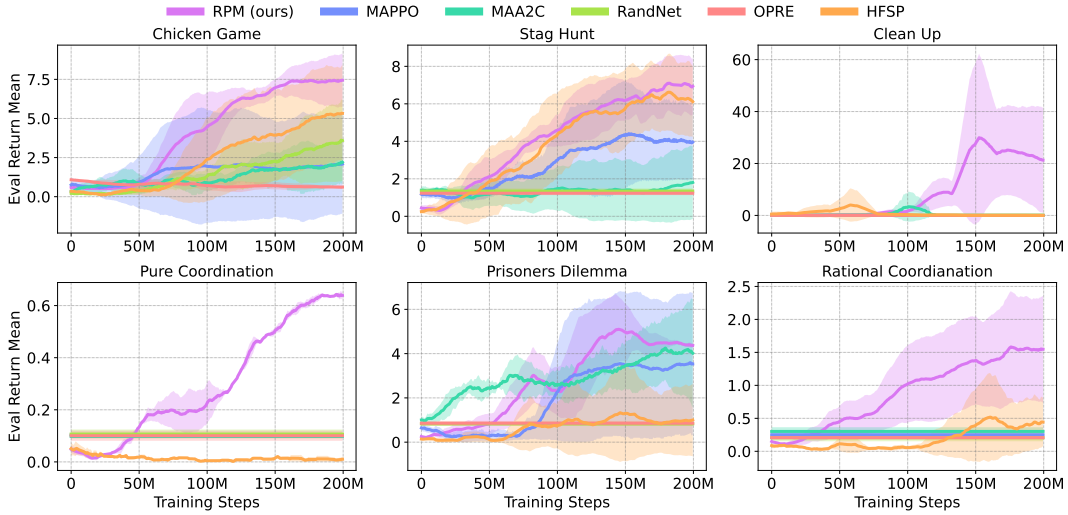

Figure 14: Episode return in substrates.

F EXTENDED 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, no technique to achieve generalization in MARL has been proposed in (Mahajan et al., 2022).

Several recent studies (Strouse et al., 2021; Lupu et al., 2021; Tang et al., 2021) have used population-based training to enhance multi-agent interaction by improving multi-agent diversity. Fictitious Co-Play (FCP) proposed in (Strouse et al., 2021) aims to learn policies for a two-agent cooperative game. It is a two-stage method. In the first stage, n agents are trained independently with different random seeds via self-play. It needs n seeds and much more extra computation. In the second stage, an FCP agent is trained by interacting with the trained policies of n agents. Another extra run of training is also needed. However, our method, RPM, is an end-to-end training method for social dilemmas, competitive, cooperative and even mixed environments with more than two agents. It needs only one run training (i.e., end-to-end one step) by utilizing fictitious self-play to sample n policies for each agent without maintaining n populations of agents. Lupu et al. (2021) considered the problem of zero-shot coordination and proposed a method to achieve diversity via population-based training (PBT) method. In contrast, our work aims to achieve the generalization of coordination, competition and social dilemmas in multi-agent systems via our novel ranked policy memory method. The proposed method in (Lupu et al., 2021) cannot be applied to competition and social dilemma scenarios. Besides that, PBT needs much more computation, which could be computationally expensive for large-scale multi-agent scenarios. Lupu et al. (2021) is also pointed out that "self-play (SP) agents control their own trajectory distribution during training, each policy typically only performs well on this exact distribution." We believe that is the key issue of the self-play method. The issue can be alleviated by introducing ranks, and each agent loads its previous policy for multi-agent self-play. The RPG (Tang et al., 2021) method is a population-based training method without self-play. It is highly dependent on the randomized reward function. For simple grid world scenarios in (Tang et al., 2021), conducting randomized reward function perturbations is not challenging. However, it is non-trivial to find proper reward function perturbations for complex scenarios.

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.

Meta-learning in MARL (Al-Shedivat et al., 2018; Kim et al., 2021) aims to address the non-stationarity issue in MARL, a well-known issue that has been extensively studied. To address the issue, the two works adopted the learning-to-learn framework. Typically, Al-Shedivat et al. (2018)

considers the problem of continuous adaptation in non-stationary environments where agents have few-shot interactions, *i.e.*, the agent must learn from only a limited amount of experience that it can collect before its environment changes. The MAML framework was used to address the issue. Kim et al. (2021) proposed a novel meta-multiagent policy gradient theorem that directly accounts for the non-stationary policy dynamics inherent to multiagent learning settings. The proposed meta-multiagent policy gradient theorem explicitly models the learning procedure of the other agent (peer learning) in two-agent settings via considering the sequential dependence of the future parameters of other agents on the meta-agent i's parameter. While the previous work (Al-Shedivat et al., 2018) ignored it. However, it would be difficult to train 3+ meta-agents simultaneously in complex scenarios due to the infinite recursion problem associated with the meta-learning framework in (Kim et al., 2021). The two works consider two-agent settings. However, solving the non-stationarity issue in scenarios where there are more than two agents is more challenging than in two-agent settings. The performance of the two works in these scenarios is yet to be investigated. Furthermore, adapting the framework of meta-learning in MARL to improve the generalization in MARL is non-trivial.