ROTATE: Regret-driven Open-ended Training for Ad Hoc Teamwork

Anonymous authors
Paper under double-blind review

Abstract

Developing AI agents capable of collaborating with previously unseen partners is a fundamental generalization challenge in multi-agent learning, known as Ad Hoc Teamwork (AHT). Existing AHT approaches often adopt a two-stage pipeline, where first, a fixed population of teammates is generated with the idea that they should be representative of the teammates that will be seen at deployment time, and second, an AHT agent is trained to collaborate well with agents in the population. To date, the research community has focused on designing separate algorithms for each stage. This separation has led to algorithms that generate teammates with limited coverage of possible behaviors, and that ignore whether the generated teammates are easy to learn from for the AHT agent. Furthermore, algorithms for training AHT agents typically treat the set of training teammates as static, thus attempting to generalize to previously unseen partner agents without assuming any control over the distribution of training teammates. This paper presents a unified framework for AHT by reformulating the problem as an open-ended learning process between an ad hoc agent and an adversarial teammate generator. We introduce ROTATE, a regret-driven, open-ended training algorithm that alternates between improving the AHT agent and generating teammates that probe its deficiencies. Extensive experiments across diverse AHT environments demonstrate that ROTATE significantly outperforms baselines at generalizing to an unseen set of evaluation teammates, thus establishing a new standard for robust and generalizable teamwork.

1 Introduction

As the deployment of AI agents in diverse applications becomes more common, it is increasingly crucial that they can collaborate effectively with previously unseen humans and other AI agents. While methods for training teams of agents have been explored in cooperative multi-agent reinforcement learning (CMARL) [18, 45], previous work [55, 42] highlights that CMARL-based agents fail to perform optimally when collaborating with unfamiliar teammates. Rather than learning strategies that are only effective against jointly trained teammates, dealing with previously unseen teammates requires adaptive AI agents that efficiently approximate the optimal strategy for collaborating with diverse teammates. Methods to train adaptive agents in the context of collaborative tasks have previously been explored within the literature of ad hoc teamwork (AHT) [6, 52, 34] and zero-shot coordination (ZSC) [21, 13, 33].

Prior work [34] has often decomposed AHT learning into two stages, consisting of first creating a fixed set of teammates, and then training an AHT agent using reinforcement learning (RL) approaches, based on interactions with teammates sampled from the set. Despite relying on neural network policies, algorithms that train an AHT agent based on interaction with a human-designed set of heuristic or pretrained agents [40, 66, 42] often struggle to handle novel behaviors outside the predefined set of teammates [53, 8]. Recent work [33, 9, 43, 44, 48] strengthens the generalization capabilities of previous methods by substituting the human-designed set of teammates with a set of teammate policies that are trained to maximize different notions of diversity. One such diversity notion is *adversarial diversity* [43, 9], which seeks to generate a set of teams that cooperate well within

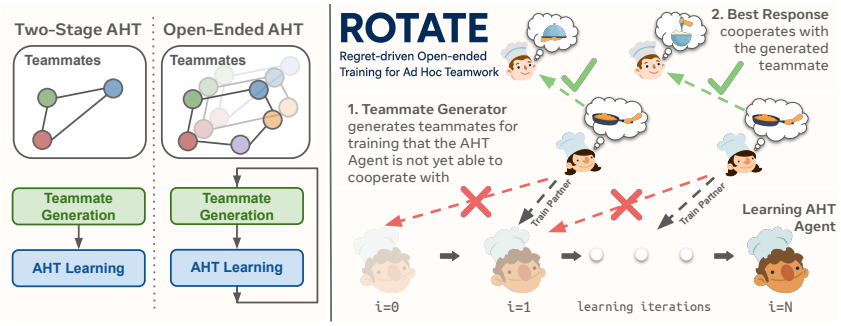


Figure 1: ROTATE *Overview.* ROTATE is an open-ended ad hoc teamwork learning framework in which the AHT agent learns as the set of training teammates expands. The core idea of ROTATE is to improve the AHT agent and iteratively generate diverse teammates with whom the AHT agent struggles to collaborate, yet not so adversarial that effective teamwork becomes impossible.

the team, but not across teams. However, prior work [14, 48, 10] empirically demonstrates that adversarial diversity often leads to teammate policies that actively diminish returns when interacting with agents other than their identified teammate, a phenomenon sometimes called *self-sabotage*.

This paper addresses two issues that cause existing AHT and ZSC methods to fail to learn policies that effectively collaborate with some teammates. First, existing methods [40, 66, 42, 43] learn from sampling teammates from a fixed set containing few teammate policies. Compared to the vast space of strategies a teammate may adopt, the AHT agent will only be trained to collaborate optimally with a small set of strategies, while potentially performing poorly against others. Second, existing methods focus on designing diverse and incompatible teammates [9, 43, 64], whose return-diminishing tendencies make it challenging for a randomly initialized, RL-based AHT agent to effectively learn to collaborate with.

In this paper, we present a fresh perspective on AHT, by observing that maximizing the return of an AHT agent on an unknown set of teammates is equivalent to minimizing its *cooperative regret*: the utility gap between the best response to a given teammate, and the AHT agent's performance with that teammate. Inspired by the success of regret for designing generally capable agents that efficiently solve a broad range of tasks [58, 15, 24, 46], we then reformulate the AHT problem as a minimax game between the AHT agent and a teammate generator. Our problem formulation suggests an open-ended framework for AHT, that drives a teammate generator to continually discover new teammate policies, while jointly improving the AHT agent. Building on this theoretical foundation, we propose a practical algorithm, ROTATE (Fig. 1), which optimizes a regret-based minimax objective for both players, while maintaining a population of all teammates explored. Key to the success of ROTATE is a novel and practical *per-state* regret objective, designed to mitigate the self-sabotage problem that naturally arises from cooperative regret type objectives. We demonstrate that ROTATE significantly improves the robustness of AHT agents when faced with previously unseen teammates, compared to a range of baselines on Level-Based Foraging and Overcooked tasks.

This paper makes three main contributions. First, it defines a novel problem formulation for AHT, enabling open-ended AHT training that continually generates new teammates. Second, it introduces a novel algorithm, ROTATE, that instantiates the proposed open-ended AHT framework. Third, it provides empirical evaluations demonstrating that ROTATE significantly improves return against unseen teammates compared to representative baselines from AHT and open-ended learning.

2 Related Work

This section provides an abridged overview of related work. The full discussion may be found in App. A. Training adaptive ego agents that may optimally coordinate with previously unseen teammates has been previously explored in AHT [52] & ZSC [21]. In particular, teammate generation approaches seek to enable generalization to unseen teammates by designing a diverse set of training teammates [53, 33]. Recent teammate generation methods [9, 43, 48] propose generating teammate policies with diverse best responses, which strongly resembles the cooperative regret objective used

by ROTATE. Our proposed method is heavily influenced by teammate generation methods, as well as prior work in open-ended learning (OEL) [23, 4], which seeks to continually design novel tasks to create a generally capable agent [58, 15, 41, 25, 24, 46].

3 Background

The interaction between agents in an AHT setting may be modeled as a decentralized, partially observable Markov decision process (Dec-POMDP) [39]. A Dec-POMDP is characterized by a 9-tuple, $\langle N, S, \{\mathcal{A}^i\}_{i=1}^{|N|}, P, p_0, R, \{\Omega^i\}_{i=1}^{|N|}, O, \gamma \rangle$, where N, S, and γ respectively denote the index set of agents within an interaction, the state space, and a discount rate in $(0, 1)$. Each interaction between the AHT agent and its teammates begins from an initial state s_0 sampled from an initial state distribution, $p_0(s)$. Denoting the set of all probability distributions over a set X as $\Delta(X)$ and the current timestep as t, a Dec-POMDP assumes that each agent may not perceive the current state, s_t. Each agent instead perceives an observation from its observation space, $o_t^i \in \Omega^i$, sampled from the observation function, $O : S \mapsto \Delta(\Omega^1 \times \cdots \times \Omega^{|N|})$. Each agent $i \in N$ then chooses an action at time t from its action space, $a_t^i \in \mathcal{A}^i$, based on a policy, $\pi^i(H_t^i)$, conditioned on its observation-action history, $H_t^i = \{o_{\leq t}^i, a_{<t}^i\}$. The actions selected by each agent are then collectively executed as the joint action, $a_t = (a_t^1, \ldots, a_t^{|N|})$. Each agent receives a common scalar reward, r_t, based on the reward function, $R : S \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^{|N|} \mapsto \mathbb{R}$. Finally, a new state s_{t+1}, is sampled according to the environment transition function, $P : S \times \mathcal{A}^1 \times \cdots \times \mathcal{A}^{|N|} \mapsto \Delta(S)$. In this paper, the notation π^{ego} refers to a trained AHT agent policy, or *ego agent*, while π^{-i} refers to the $N-1$ policies of the AHT agent's teammates. Importantly, we assume that teammates choose their actions only based on the current state. At the same time, the AHT agent selects its actions based on its state-action history, which is necessary to allow the AHT agent to distinguish between different types of teammates effectively.

4 Ad Hoc Teamwork Problem Formulation

Ad Hoc Teamwork (AHT) methods aim to train an adaptive AHT policy that an ego agent can follow to achieve maximal return when collaborating with an unknown set of evaluation teammates. Using the Dec-POMDP formulation to model the interaction between agents, this section formalizes the objective of AHT. While the most general AHT setting considers a possibly varying number of ego agents and teammates within an interaction [57, 42], this formalization addresses the more straightforward case where there is only a single ego agent within a team.

Let π^{-i} denote a joint teammate policy controlling the $N-1$ non-ego agents during collaboration. Denote the returns of an ego agent following π^{ego} to collaborate with teammates controlled by π^{-i}, starting from state s, as:

$$V(s|\pi^{-i}, \pi^{\text{ego}}) = \mathbb{E}_{\substack{a_t^{\text{ego}} \sim \pi^{\text{ego}}, \\ a_t^{-i} \sim \pi^{-i}, P, O}} \left[\sum_{t=0}^{\infty} \gamma^t R(s_t, a_t) \middle| s_0 = s \right]. \tag{1}$$

Let Π^{eval} denote the unknown set of joint teammate policies encountered during evaluation, which is assumed to only contain competent and non-adversarial policies, as defined in the seminal work of Stone et al. [52]. Let $\psi^{\text{eval}}(\Pi^{\text{eval}})$ denote the probability distribution over Π^{eval} defining how teammates are sampled during evaluation. An ego agent policy, π^{ego}, is evaluated by its ability to maximize the expected returns when collaborating with joint teammate policies sampled from $\psi^{\text{eval}}(\Pi^{\text{eval}})$, which is formalized as:

$$\max_{\pi^{\text{ego}}} V(\psi^{\text{eval}}, \Pi^{\text{eval}}, \pi^{\text{ego}}) = \max_{\pi^{\text{ego}}} \mathbb{E}_{\pi^{-i} \sim \psi^{\text{eval}}(\Pi^{\text{eval}}), s_0 \sim p_0} \left[V(s_0|\pi^{-i}, \pi^{\text{ego}}) \right]. \tag{2}$$

An optimal π^{ego} that maximizes Eq. 2 closely approximates the *best response policy* performance when collaborating with $\pi^{-i} \in \Pi^{\text{eval}}$. Given a teammate policy π^{-i}, $\text{BR}(\pi^{-i})$ is a best response policy to π^{-i} if and only the team policy formed by π^{-i} and $\text{BR}(\pi^{-i})$ achieves maximal return:

$$\text{BR}(\pi^{-i}) := \max_{\pi} \mathbb{E}_{s \sim p_0} \left[V(s|\pi, \pi^{-i}) \right]. \tag{3}$$

In some cases, an AHT algorithm can directly estimate this optimal policy by using Π^{eval} to train an ego agent policy that maximizes $V(\psi^{\text{eval}}, \Pi^{\text{eval}}, \pi^{\text{ego}})$ when Π^{eval} is known.[1] However, most AHT methods address the more challenging case where Π^{eval} is unknown, which is the setting that this paper adopts as well. While our methods assume no knowledge of Π^{eval} during training, we follow standard practice [40, 42, 66, 57] by manually designing a diverse Π^{eval} for evaluation purposes, as we later describe in Section 7.

When Π^{eval} is unknown, AHT algorithms [34] either assume access to a training set of teammate policies, Π^{train}, or generate such a set. An expert's domain knowledge about the characteristics of Π^{eval} may be leveraged to construct a Π^{train} similar to Π^{eval}. Once the set of training teammates has been formed, current AHT algorithms use reinforcement learning to discover an ego agent policy based on interactions with joint policies sampled from Π^{train}. While the precise training objective varies with the AHT algorithm, a common objective is to maximize the expected return during interactions with joint policies sampled uniformly from Π^{train}:

$$\pi^{*,\text{ego}}(\Pi^{\text{train}}) = \underset{\pi^{\text{ego}}}{\text{argmax}}\, \mathbb{E}_{\pi^{-i}\sim\mathcal{U}(\Pi^{\text{train}}), s_0\sim p_0}\left[V(s_0|\pi^{-i}, \pi^{\text{ego}})\right]. \tag{4}$$

Naturally, even an optimal ego agent policy, $\pi^{*,\text{ego}}(\Pi^{\text{train}})$, may not be optimal with respect to Π^{eval} and ψ^{eval}, due to the potential distribution shift caused by differences between the training and evaluation objectives.

5 An Open-Ended Learning Perspective on Ad Hoc Teamwork

In this section, we outline the general components of our open-ended framework to train ego agents that are performant at collaborating with holdout teammate policies, despite not knowing Π^{eval} and ψ^{eval} during training. We first argue for minimizing worst-case cooperative regret towards training ego agent policies that maximize Eq. 2 when Π^{eval} is unknown. We then finish the section by introducing two necessary procedures in an iterative process to minimize worst-case regret.

We define the *cooperative regret* of an ego agent policy π^{ego} when interacting with some joint teammate policy π^{-i} from a starting state s as:

$$\text{CR}(\pi^{\text{ego}}, \pi^{-i}, s) = V\left(s|\pi^{-i},\ \text{BR}(\pi^{-i})\right) - V\left(s|\pi^{-i}, \pi^{\text{ego}}\right). \tag{5}$$

Any optimal AHT policy that maximizes Eq. 2 must also minimize the expected regret over joint teammate policies sampled based on $\psi^{\text{eval}}(\Pi^{\text{eval}})$, which we formally express as:

$$\text{CR}(\psi^{\text{eval}}, \Pi^{\text{eval}}, \pi^{\text{ego}}) = \mathbb{E}_{\pi^{-i}\sim\psi^{\text{eval}}(\Pi^{\text{eval}}), s_0\sim p_0}\left[\text{CR}((\pi^{\text{ego}}, \pi^{-i}, s_0)\right]. \tag{6}$$

This property is a consequence of $V\left(s|\pi^{-i},\ \text{BR}(\pi^{-i})\right)$ being independent of π^{ego} for any π^{-i} and s, leaving maximizing expected regret equivalent to minimizing the negative expected returns when collaborating with joint teammate policies sampled from $\psi^{\text{eval}}(\Pi^{\text{eval}})$.

Without knowing Π^{eval} to optimize $\text{CR}(\psi^{\text{eval}}, \Pi^{\text{eval}}, \pi^{\text{ego}})$, we instead take inspiration from approaches in UED [58, 15], and propose optimizing π^{ego} to minimize the *worst-case regret* that could be induced by any teammate policy π^{-i}:

$$\min_{\pi^{\text{ego}}} \max_{\pi^{-i}\in\Pi^{-i}} \mathbb{E}_{s_0\sim p_0}\left[\text{CR}(\pi^{\text{ego}}, \pi^{-i}, s_0)\right], \tag{7}$$

where we re-emphasize that Π^{-i} denotes the set of all competent and non-adversarial [52] joint teammate policies. Limiting the considered joint policies is important since teams that always perform poorly against any good-faith π^{ego} are unlikely to be encountered in coordination scenarios and may induce unnecessary learning challenges for RL-based AHT learning algorithms.

Finding π^{ego} that achieves zero worst-case regret is equivalent to finding an ego agent that achieves the best-response return with any joint teammate policy π^{-i}. If such a π^{ego} exists, then this AHT

[1] In the context of reinforcement-learning-based AHT algorithms, "known" means that an AHT algorithm has unlimited sampling access to the teammate policies.

agent would maximize Eq. 2 for any ψ^{eval} and Π^{eval}—however, existence is not guaranteed [30]. In practice, we are content with *minimizing* the worst-case regret. While minimizing worst-case regret still applies to AHT problems with more than one teammate at a time, note that we limit our method for optimizing Eq. 7 and our experiments to two-player, fully observable AHT games.

Algorithm 1 in the Appendix outlines a framework to train a π^{ego} that minimizes worst-case regret. The algorithm resembles coordinate ascent algorithms [16], which alternate between optimizing for π^{-i} and π^{ego} for T iterations, while assuming the other is fixed. We call a phase where we fix π^{ego} and update π^{-i} to maximize the ego agent's regret, the *teammate generation phase*. Meanwhile, assuming that π^{-i} is fixed, the *ego agent update phase* updates π^{ego} to minimize regret. This optimization algorithm is an open-ended training method that continually generates novel teammate policies whose interaction with the ego agent provides the learning experience to improve π^{ego}. Next, we detail the learning process during these two phases.

6 ROTATE: Regret-driven Open-ended Training for Ad hoc TEamwork

This section presents our regret-driven, open-ended AHT algorithm, ROTATE. We first describe the teammate generation procedure in Section 6.1, particularly focusing on motivating the objective we used to generate teammate policies. Next, we provide details of the ego agent update method in Section 6.2. The Appendix provides ROTATE's pseudocode and additional implementation details.

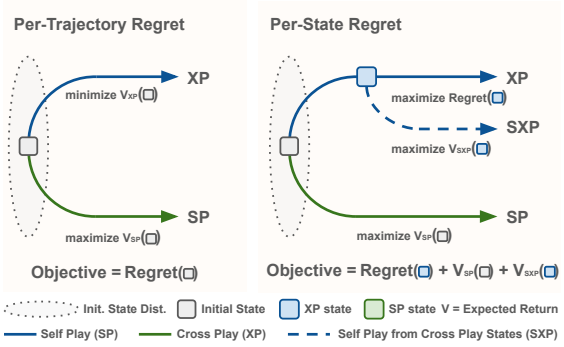


Figure 2: Per-trajectory regret vs per-state regret.

6.1 ROTATE Teammate Generator

The teammate generator produces teammate policies that maximize the regret of π^{ego}. By updating π^{ego} to minimize its regret against the regret-maximizing teammate policy, we aim to decrease the worst-case cooperative regret of π^{ego} (Eq. 7). Since measuring cooperative regret requires estimating the performance of a generated π^{-i} when collaborating with $\text{BR}(\pi^{-i})$, we jointly train policies for π^{-i} and an approximation of $\text{BR}(\pi^{-i})$ using the PPO algorithm [50].

Before detailing our alternative objectives for training π^{-i}, we first introduce the different interactions that provide the experience to train π^{-i}. Let self-play (SP) refer to teammate and best response interactions, cross-play (XP) refer to teammate and ego agent interactions, and cross-play continued by self-play (SXP) refer to an interaction where the teammate is first interacting with the ego agent, but switches at a random timestep t to interacting with the best response. We train π^{-i} based on states sampled from SP, XP, and SXP. Let $d(\pi^1, \pi^2; p_0)$ denote the state visitation distribution when π^1 and π^2 interact based on a starting state distribution p_0. To denote the state visitation distributions for these interactions, we use the following shorthand:

$$p_{\text{SP}} := d\left(\pi^{-i}, \text{BR}(\pi^{-i}); p_0\right), \quad p_{\text{XP}} := d\left(\pi^{-i}, \pi^{\text{ego}}; p_0\right), \quad p_{\text{SXP}} := d\left(\pi^{-i}, \text{BR}(\pi^{-i}); p_{\text{XP}}\right). \tag{8}$$

This section considers two teammate policy generation objectives that differ in the data source used to optimize the objective, and are illustrated in Fig. 2.

The first objective generates π^{-i} based on maximizing *per-trajectory regret* of π^{ego}, which only maximizes regret of trajectories starting from the initial state distribution:

$$\max_{\pi^{-i}} \mathbb{E}_{s_0 \sim p_0}\left[\text{CR}(\pi^{\text{ego}}, \pi^{-i}, s_0)\right]. \tag{9}$$

From the definition of CR (Eq. 5), optimizing per-trajectory regret amounts to maximizing the expected returns of SP interactions from the initial state distribution p_0, $\mathbb{E}_{s \sim p_0}\left[V(s|\pi^{-i}, \text{BR}(\pi^{-i}))\right]$,

and minimizing the expected returns of XP interactions from p_0, $\mathbb{E}_{s\sim p_0}\left[V(s|\pi^{-i}, \pi^{\text{ego}})\right]$. This objective resembles the adversarial diversity metric optimized in prior teammate generation work [9, 43, 14].

While the per-trajectory regret is the same as the regret objective optimized in Eq. 7, optimizing it naively leads to generating teammates with undesirable self-sabotage behaviors. To minimize the expected returns from cross-play interaction, self-sabotaging policies choose actions leading to low returns in states outside the support of p_{SP} [14], including the states visited during the collaboration between π^{-i} and π^{ego}, $s \sim p_{\text{XP}}$. The lack of high reward signals makes it challenging for the ego agent to learn to collaborate with π^{-i} using reinforcement learning.

The second objective discourages the emergence of self-sabotaging policies by optimizing a *per-state regret* objective defined as:

$$\max_{\pi^{-i}} \left(\mathbb{E}_{s\sim p_{\text{XP}}}[\text{CR}(\pi^{\text{ego}}, \pi^{-i}, s)] + \mathbb{E}_{s\sim p_{\text{SXP}}}[V(s|\pi^{-i}, \text{BR}(\pi^{-i}))] + \mathbb{E}_{s\sim p_{\text{SP}}}[V(s|\pi^{-i}, \text{BR}(\pi^{-i}))]\right). \tag{10}$$

The first term in Expr. 10 encourages discovering π^{-i} for which the ego agent policy has a high room for improvement. Meanwhile, the second term trains π^{-i} to act as if it interacts with its best response policy, even from starting states encountered during XP interactions. This encourages π^{-i} to act in *good faith* by having at least a partner policy that can collaborate well with π^{-i} when starting from states in p_{XP}. Finally, the last term encourages π^{-i} to cooperate with its best response. This enables consistently generating competent teammates during open-ended learning, which is essential as stated in Section 5.

While obtaining states from p_{SP} and p_{XP} is straightforward, states from p_{SXP} can be tricky to collect depending on the implementation of an AHT environment. If an environment supports resetting to arbitrary states, then states encountered during XP interaction can be stored and used as the initial state for SP interactions. Otherwise, we can use a data collection strategy that first samples a random timestep t, runs XP interaction until timestep t, and finally switches to SP interaction afterwards [48]. Only data gathered after timestep t should be used to compute objectives based on p_{SXP}.

6.2 ROTATE Ego Agent Update

At each iteration, ROTATE creates a teammate that attempts to discover cooperative weaknesses of the previous iteration's ego agent, by maximizing its per-state regret. To allow the ROTATE ego agent to improve its robustness over time and reduce the possibility that it forgets how to cope with previously generated teammates, the ROTATE ego agent maintains a *population buffer* of generated teammates. During the ego agent update phase of each iteration, the ROTATE ego agent is trained using PPO [50] against teammates sampled uniformly from the population buffer. We find that for the ego agent to learn effectively against the nonstationary population buffer, it is important to define a lower entropy coefficient and learning rate than when training the teammate and BR agents (in the range of 1×10^{-4} for the entropy coefficient and 1×10^{-5} for the learning rate).

7 Experimental Results

This section presents the empirical evaluation of ROTATE compared to baseline methods, across six cooperative tasks. The main research questions are:

- **RQ1**: Does ROTATE better generalize to unseen teammates, compared to baseline methods from the AHT and UED literature? (Yes)
- **RQ2**: Does per-state regret improve over trajectory level regret and mixed-play regret? (Yes)
- **RQ3**: Is the population buffer necessary for ROTATE to learn well? (Yes)
- **RQ4**: Is the ROTATE population useful for training an independent ego agent? (Yes)

We first describe the experimental setting, including tasks, baselines, construction of the evaluation set, and the evaluation metric, followed by presenting results in Sec. 7.1.

Tasks ROTATE is evaluated on six tasks: Level-Based Foraging (LBF) [2], and five layouts from the Overcooked suite [8]: Cramped Room (CR), Asymmetric Advantages (AA), Counter Circuit (CC), Coordination Ring (CoR), and Forced Coordination (FC). All six tasks are fully cooperative with a variety of possible coordination conventions, and are commonly used within the AHT literature [2, 12, 40]. In our LBF configuration, two agents must navigate to three apples that are randomly placed within a gridworld, and cooperate to pick up the apples. In all Overcooked tasks, two agents collaborate in a gridworld kitchen to cook and deliver onion soup, where the main difference between tasks is the kitchen layout. All experiments were implemented with JAX [7], so we use JAX re-implementations of the LBF and Overcooked tasks [5, 47].

Baselines As our method is most closely related to methods from unsupervised environment design (UED) and teammate generation, we compare against two UED methods adapted for AHT (PAIRED [15], Minimax Return [36, 56]) and three teammate generation methods (Fictitious Co-Play [53], BRDiv [43], CoMeDi [48]). While curator-based methods such as PLR [24, 25] are popular in UED, we do not compare against them as they are orthogonal to ROTATE [17, 56, 11]. Similarly, we do not compare against AHT algorithms that propose techniques to improve ego learning [3]. Each baseline is described in App. C, along with implementation details. For fair comparison, all open-ended and UED methods were trained for a similar number of environment interactions, or until best performance on the evaluation set. All teammate generation approaches were ran using a similar number of environment interactions as their original implementations, as scaling them up to use a similar number of steps as the open-ended approaches proved challenging (see discussion in App. C). All results are reported with three trials.

Construction of Π^{eval} We wish to evaluate all methods on as diverse a set of evaluation teammates as practically feasible, while ensuring that each teammate acts in "good faith". To achieve this goal, for each task, we construct 9 to 13 evaluation teammates using three methods: IPPO with varied seeds and reward shaping, BRDiv, and manually programmed heuristic agents. Full descriptions of the teammate generation procedure and all teammates in Π^{eval} are provided in App. H.

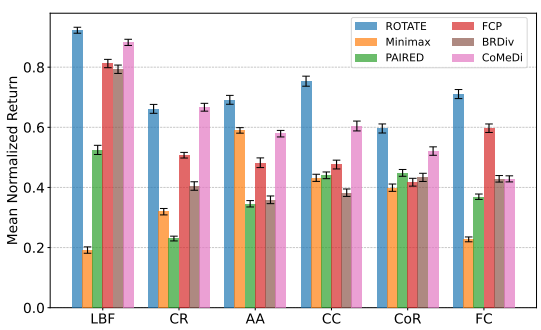
Evaluation Metric Ego agent policies are evaluated with each teammate in Π^{eval} for 64 evaluation episodes, where the return is computed for each episode, and normalized using a lower return bound of zero and an estimated best response return as the upper bound for each teammate. Performance of a method on Π^{eval} is reported as the normalized mean return with bootstrapped 95% confidence intervals, computed via the `rliable` library [1]. Details about the normalization procedure and specific bounds for each teammate are reported in App. H.

7.1 Results

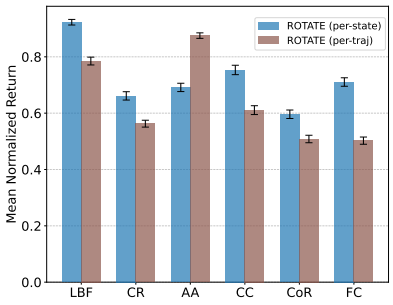
This section presents empirical analysis addressing RQ1 and RQ2, while analysis addressing RQ3 and RQ4 are provided in App. D. Supplemental analysis considering alternative regret-based objectives, breaking down performance by evaluation teammate type, and learning curves for all variants of ROTATE are provided in App. E.

RQ1: Does ROTATE better generalize to unseen teammates compared to baselines? (Yes) To evaluate the generalization capabilities of ROTATE, we compare its performance against baselines on Π^{eval}. Fig. 3a compares the normalized mean returns for ROTATE and all baseline methods across the six tasks, showing that ROTATE significantly outperforms all baselines on 5/6 tasks.

Among the baseline methods, the next best performing baselines are CoMeDi and FCP. We attribute CoMeDi's strong performance to the resemblance of its mixed-play objective to our per-state regret objective, which we discuss in App. E.1. FCP's performance may be attributed to the large number of partners that FCP was trained with (approximately 100 teammates per task). We found that FCP tends to perform especially well with the IPPO policies in Π^{eval}, likely because the IPPO evaluation teammates are in-distribution for the distribution of teammates constructed by FCP. We also observe that Minimax Return performs surprisingly well in AA, which we hypothesize is due to AA's particular characteristics. In AA, agents operate in separated kitchen halves, possessing all necessary resources for individual task completion, with shared access to pots on the dividing counter being the only shared resource. Consequently, a fully adversarial partner has limited methods to sabotage



(a) ROTATE vs baselines.



(b) ROTATE with per-state regret (ours) vs ROTATE with per-trajectory regret.

Figure 3: (Left) ROTATE outperforms all baseline methods across all tasks in evaluation return. (Right) ROTATE with per-state regret (ours) outperforms ROTATE with per-trajectory regret in $5/6$ tasks. 95% bootstrapped CI's are shown, computed across all evaluation teammates and trials.

the ego agent.[2] However, on LBF and FC, where coordination is crucial to obtain positive return on the tasks, Minimax Return is the worst performing baseline.

BRDiv and PAIRED exhibit comparatively poor performance, which may be partially attributed to their teammate generation objectives that resemble per-trajectory regret. As we find for **RQ2**, per-state regret outperforms per-trajectory regret within the ROTATE framework. Furthermore, PAIRED's update structure involves a lockstep training process for the teammate generator, best response, and ego agent. This synchronized training may hinder the natural emergence of robust conventions that are crucial for effective AHT.

RQ2: Does per-state regret improve over trajectory regret? (Yes) As discussed in Section 6, we propose that teammates should maximize per-state regret rather than per-trajectory regret to mitigate the emergence of self-sabotage behaviors. Here, we compare ROTATE where the teammate maximizes per-state regret (ours) to ROTATE where the teammate maximizes per-trajectory regret. All configurations other than the teammate's policy objective are kept identical, including the data used to train the teammate value functions. Fig. 3b shows that ROTATE with per-state regret outperforms ROTATE with per-trajectory regret on all tasks except AA, confirming the superiority of per-state regret. As discussed in **RQ1**, we observe that AA is a layout where an ego agent is less susceptible to sabotage, due to the separated kitchen layout. App. E.2 presents additional experiments testing ROTATE with CoMeDi-style mixed-play rollouts, and alternative methods to compute per-state regret—ultimately finding that ROTATE outperforms all variations.

8 Conclusion

This paper reformulates the AHT problem as an open-ended learning problem and introduces ROTATE, a regret-driven algorithm. ROTATE iteratively alternates between improving an AHT agent and generating challenging yet cooperative teammates by optimizing a per-state regret objective designed to discover teammates that exploit cooperative vulnerabilities without encouraging self-sabotage. Empirical evaluations across six AHT tasks demonstrate that ROTATE significantly enhances the generalization capabilities of AHT agents when faced with previously unseen teammates, outperforming a range of baselines from both AHT and UED. The current work has several limitations that future work may address. First, the paper only studies ROTATE on two-agent, fully observable, and fully cooperative scenarios. Second, this work has focused on the teammate generation phase of open-ended AHT. Future work might explore ego agent training methods that better handle the nonstationarity induced by open-ended teammate generation.

[2]Agent teams may still achieve higher returns through effective coordination on AA due to layout asymmetry. In the "left" kitchen, the delivery zone is adjacent to the pots while the onions are farther, while in the "right" kitchen, the opposite is true. An efficient team has the "left" agent delivering finished soup, and the "right" agent placing onions in the pots.

References

[1] Rishabh Agarwal, Max Schwarzer, Pablo Samuel Castro, Aaron C Courville, and Marc Bellemare. Deep Reinforcement Learning at the Edge of the Statistical Precipice. In *Advances in Neural Information Processing Systems*, volume 34, pp. 29304–29320. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/hash/f514cec81cb148559cf475e7426eed5e-Abstract.html.

[2] Stefano V. Albrecht and Subramanian Ramamoorthy. A game-theoretic model and best-response learning method for ad hoc coordination in multiagent systems. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, AAMAS '13, pp. 1155–1156, Richland, SC, May 2013. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-1993-5.

[3] Stefano V. Albrecht and Peter Stone. Autonomous agents modelling other agents: A comprehensive survey and open problems. *Artificial Intelligence*, 258:66–95, 2018. ISSN 0004-3702. DOI: https://doi.org/10.1016/j.artint.2018.01.002. URL https://www.sciencedirect.com/science/article/pii/S0004370218300249.

[4] 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*, September 2019. URL https://openreview.net/forum?id=SkxpxJBKwS.

[5] Clément Bonnet, Daniel Luo, Donal John Byrne, Shikha Surana, Sasha Abramowitz, Paul Duckworth, Vincent Coyette, Laurence Illing Midgley, Elshadai Tegegn, Tristan Kalloniatis, Omayma Mahjoub, Matthew Macfarlane, Andries Petrus Smit, Nathan Grinsztajn, Raphael Boige, Cemlyn Neil Waters, Mohamed Ali Ali Mimouni, Ulrich Armel Mbou Sob, Ruan John de Kock, Siddarth Singh, Daniel Furelos-Blanco, Victor Le, Arnu Pretorius, and Alexandre Laterre. Jumanji: a Diverse Suite of Scalable Reinforcement Learning Environments in JAX. In *The Twelfth International Conference on Learning Representations*, October 2023. URL https://openreview.net/forum?id=C4CxQmp9wc.

[6] Michael Bowling and Peter McCracken. Coordination and adaptation in impromptu teams. In *AAAI*, volume 5, pp. 53–58, 2005.

[7] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

[8] Micah Carroll, Rohin Shah, Mark K Ho, Tom Griffiths, Sanjit Seshia, Pieter Abbeel, and Anca Dragan. On the Utility of Learning about Humans for Human-AI Coordination. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/hash/f5b1b89d98b7286673128a5fb112cb9a-Abstract.html.

[9] Rujikorn Charakorn, Poramate Manoonpong, and Nat Dilokthanakul. Generating diverse cooperative agents by learning incompatible policies. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=UkU05GOH7_6.

[10] Rujikorn Charakorn, Poramate Manoonpong, and Nat Dilokthanakul. Diversity is not all you need: Training a robust cooperative agent needs specialist partners. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 56401–56423. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/66b35d2e8d524706f39cc21f5337b002-Paper-Conference.pdf.

[11] Paresh Chaudhary, Yancheng Liang, Daphne Chen, Simon S. Du, and Natasha Jaques. Improving human-ai coordination through adversarial training and generative models, 2025. URL https://arxiv.org/abs/2504.15457.

[12] Filippos Christianos, Lukas Schäfer, and Stefano V Albrecht. Shared experience actor-critic for multi-agent reinforcement learning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2020.

[13] Brandon Cui, Hengyuan Hu, Luis Pineda, and Jakob Foerster. K-level reasoning for zero-shot coordination in hanabi. *Advances in Neural Information Processing Systems*, 34:8215–8228, 2021.

[14] Brandon Cui, Andrei Lupu, Samuel Sokota, Hengyuan Hu, David J Wu, and Jakob Nicolaus Foerster. Adversarial diversity in hanabi. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=uLE3WF3-H_5.

[15] Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent Complexity and Zero-shot Transfer via Unsupervised Environment Design. In *Advances in Neural Information Processing Systems*, volume 33, pp. 13049–13061. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/hash/985e9a46e10005356bbaf194249f6856-Abstract.html.

[16] DA d'Esopo. A convex programming procedure. *Naval Research Logistics Quarterly*, 6(1): 33–42, 1959.

[17] Hannah Erlebach and Jonathan Cook. RACCOON: Regret-based adaptive curricula for cooperation. In *Coordination and Cooperation for Multi-Agent Reinforcement Learning Methods Workshop*, 2024. URL https://openreview.net/forum?id=jAH5JNY3Qd.

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

[19] Peter W Glynn. Likelihood ratio gradient estimation for stochastic systems. *Communications of the ACM*, 33(10):75–84, 1990.

[20] Pengjie Gu, Mengchen Zhao, Jianye Hao, and Bo An. Online ad hoc teamwork under partial observability. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=18Ys0-PzyPI.

[21] Hengyuan Hu, Adam Lerer, Alex Peysakhovich, and Jakob Foerster. "other-play" for zero-shot coordination. In *International Conference on Machine Learning*, pp. 4399–4410. PMLR, 2020.

[22] Hengyuan Hu, Adam Lerer, Brandon Cui, Luis Pineda, Noam Brown, and Jakob Foerster. Off-belief learning. In *International Conference on Machine Learning*, pp. 4369–4379. PMLR, 2021.

[23] Edward Hughes, Michael D. Dennis, Jack Parker-Holder, Feryal Behbahani, Aditi Mavalankar, Yuge Shi, Tom Schaul, and Tim Rocktäschel. Position: Open-Endedness is Essential for Artificial Superhuman Intelligence. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 20597–20616. PMLR, July 2024. URL https://proceedings.mlr.press/v235/hughes24a.html.

[24] Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. *Advances in Neural Information Processing Systems*, 34:1884–1897, 2021.

[25] Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.

[26] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. In *3rd International Conference for Learning Representations*, San Diego, CA, 2015. DOI: 10.48550/arXiv.1412.6980. URL http://arxiv.org/abs/1412.6980. arXiv:1412.6980 [cs].

[27] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. Overcoming catastrophic forgetting in neural networks. *Proceedings of the National Academy of Sciences*, 114(13):3521–3526, March 2017. DOI: 10.1073/pnas.1611835114. URL https://www.pnas.org/doi/10.1073/pnas.1611835114. Publisher: Proceedings of the National Academy of Sciences.

[28] WB Langdon. Pfeiffer–a distributed open-ended evolutionary system. In *AISB*, volume 5, pp. 7–13. Citeseer, 2005.

[29] Fanqi Lin, Shiyu Huang, Tim Pearce, Wenze Chen, and Wei-Wei Tu. TiZero: Mastering Multi-Agent Football with Curriculum Learning and Self-Play. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, AAMAS '23, pp. 67–76, Richland, SC, May 2023. International Foundation for Autonomous Agents and Multiagent Systems. ISBN 978-1-4503-9432-1.

[30] Robert Loftin and Frans A Oliehoek. On the impossibility of learning to cooperate with adaptive partner strategies in repeated games. In *International Conference on Machine Learning*, pp. 14197–14209. PMLR, 2022.

[31] Chris Lu, Yannick Schroecker, Albert Gu, Emilio Parisotto, Jakob Nicolaus Foerster, Satinder Singh, and Feryal Behbahani. Structured State Space Models for In-Context Reinforcement Learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, November 2023. URL https://openreview.net/forum?id=4W9FVg1j6I&noteId=38Anv4M4TW.

[32] Keane Lucas and Ross E Allen. Any-play: An intrinsic augmentation for zero-shot coordination. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*, pp. 853–861, 2022.

[33] Andrei Lupu, Brandon Cui, Hengyuan Hu, and Jakob Foerster. Trajectory diversity for zero-shot coordination. In *International conference on machine learning*, pp. 7204–7213. PMLR, 2021.

[34] Reuth Mirsky, Ignacio Carlucho, Arrasy Rahman, Elliot Fosong, William Macke, Mohan Sridharan, Peter Stone, and Stefano V Albrecht. A survey of ad hoc teamwork research. In *European conference on multi-agent systems*, pp. 275–293. Springer, 2022.

[35] 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.

[36] Jun Morimoto and Kenji Doya. Robust Reinforcement Learning. *Neural Comput.*, 17(2):335–359, February 2005. ISSN 0899-7667. DOI: 10.1162/0899766053011528. URL https://doi.org/10.1162/0899766053011528.

[37] Hadi Nekoei, Akilesh Badrinaaraayanan, Aaron Courville, and Sarath Chandar. Continuous Coordination As a Realistic Scenario for Lifelong Learning. In *Proceedings of the 38th International Conference on Machine Learning*, pp. 8016–8024. PMLR, July 2021. URL https://proceedings.mlr.press/v139/nekoei21a.html. ISSN: 2640-3498.

[38] Hadi Nekoei, Xutong Zhao, Janarthanan Rajendran, Miao Liu, and Sarath Chandar. Towards Few-shot Coordination: Revisiting Ad-hoc Teamplay Challenge In the Game of Hanabi. In *Proceedings of The 2nd Conference on Lifelong Learning Agents*, pp. 861–877. PMLR, November 2023. URL https://proceedings.mlr.press/v232/nekoei23b.html. ISSN: 2640-3498.

[39] Frans A. Oliehoek and Christopher Amato. *A Concise Introduction to Decentralized POMDPs*. Springer Publishing Company, Incorporated, 1st edition, 2016. ISBN 3319289276.

[40] Georgios Papoudakis, Filippos Christianos, and Stefano V. Albrecht. Agent modelling under partial observability for deep reinforcement learning. In *Advances in Neural Information Processing Systems*, 2021.

[41] Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pp. 17473–17498. PMLR, 2022.

[42] Arrasy Rahman, Niklas Höpner, Filippos Christianos, and Stefano V. Albrecht. Towards Open Ad Hoc Teamwork Using Graph-based Policy Learning. In *Proceedings of the 38 th International Conference on Machine Learning*, volume 139. PMLR, June 2021.

[43] Arrasy Rahman, Elliot Fosong, Ignacio Carlucho, and Stefano V Albrecht. Generating teammates for training robust ad hoc teamwork agents via best-response diversity. *Transactions on Machine Learning Research*, 2023. ISSN 2835-8856. URL https://openreview.net/forum?id=l5BzfQhROl.

[44] Muhammad Rahman, Jiaxun Cui, and Peter Stone. Minimum coverage sets for training robust ad hoc teamwork agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17523–17530, 2024.

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

[46] Alexander Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Nicolaus Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=iEeiZlTbts.

[47] Alexander Rutherford, Benjamin Ellis, Matteo Gallici, Jonathan Cook, Andrei Lupu, Garðar Ingvarsson, Timon Willi, Ravi Hammond, Akbir Khan, Christian S. de Witt, Alexandra Souly, Saptarashmi Bandyopadhyay, Mikayel Samvelyan, Minqi Jiang, Robert Lange, Shimon Whiteson, Bruno Lacerda, Nick Hawes, Tim Rocktäschel, Chris Lu, and Jakob Foerster. JaxMARL: Multi-Agent RL Environments and Algorithms in JAX. In *Advances in Neural Information Processing Systems*, volume 37, pp. 50925–50951, December 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/5aee125f052c90e326dcf6f380df94f6-Abstract-Datasets_and_Benchmarks_Track.html.

[48] Bidipta Sarkar, Andy Shih, and Dorsa Sadigh. Diverse conventions for human-AI collaboration. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=MljeRycu9s.

[49] John Schulman, Philipp Moritz, Sergey Levine, Michael I. Jordan, and P. Abbeel. High-dimensional continuous control using generalized advantage estimation. *CoRR*, abs/1506.02438, 2015. URL https://api.semanticscholar.org/CorpusID:3075448.

[50] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *ArXiv*, abs/1707.06347, 2017. URL https://api.semanticscholar.org/CorpusID:28695052.

[51] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529 (7587):484–489, 2016.

[52] Peter Stone, Gal Kaminka, Sarit Kraus, and Jeffrey Rosenschein. Ad Hoc Autonomous Agent Teams: Collaboration without Pre-Coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, pp. 1504–1509, July 2010. DOI: 10.1609/aaai.v24i1.7529. URL https://ojs.aaai.org/index.php/AAAI/article/view/7529.

[53] DJ Strouse, Kevin McKee, Matt Botvinick, Edward Hughes, and Richard Everett. Collaborating with Humans without Human Data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P. S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 14502–14515, 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/797134c3e42371bb4979a462eb2f042a-Paper.pdf.

[54] Tim Taylor. Evolutionary innovations and where to find them: Routes to open-ended evolution in natural and artificial systems. *Artificial life*, 25(2):207–224, 2019.

[55] 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.

[56] Victor Villin, Thomas Kleine Buening, and Christos Dimitrakakis. A minimax approach to ad hoc teamwork, 2025. URL https://arxiv.org/abs/2502.02377.

[57] Caroline Wang, Arrasy Rahman, Ishan Durugkar, Elad Liebman, and Peter Stone. N-agent ad hoc teamwork. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=q7TxGUWlhD.

[58] Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International conference on machine learning*, pp. 9940–9951. PMLR, 2020.

[59] Xihuai Wang, Shao Zhang, Wenhao Zhang, Wentao Dong, Jingxiao Chen, Ying Wen, and Weinan Zhang. ZSC-Eval: An Evaluation Toolkit and Benchmark for Multi-agent Zero-shot Coordination. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, November 2024. URL https://openreview.net/forum?id=9aXjIBLwKc#discussion.

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

[61] Dong Xing, Qianhui Liu, Qian Zheng, and Gang Pan. Learning with generated teammates to achieve type-free ad-hoc teamwork. In Zhi-Hua Zhou (ed.), *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21*, pp. 472–478. International Joint Conferences on Artificial Intelligence Organization, 8 2021. DOI: 10.24963/ijcai.2021/66. URL https://doi.org/10.24963/ijcai.2021/66. Main Track.

[62] Xue Yan, Jiaxian Guo, Xingzhou Lou, Jun Wang, Haifeng Zhang, and Yali Du. An efficient end-to-end training approach for zero-shot human-AI coordination. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=6ePsuwXUwf.

[63] Chao Yu, Akash Velu, Eugene Vinitsky, Jiaxuan Gao, Yu Wang, Alexandre Bayen, and Yi Wu. The Surprising Effectiveness of PPO in Cooperative Multi-Agent Games. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24611–24624, December 2022. URL https://proceedings.neurips.cc/paper_files/paper/2022/hash/9c1535a02f0ce079433344e14d910597-Abstract-Datasets_and_Benchmarks.html.

[64] Lei Yuan, Lihe Li, Ziqian Zhang, Feng Chen, Tianyi Zhang, Cong Guan, Yang Yu, and Zhi-Hua Zhou. Learning to coordinate with anyone. In *Proceedings of the Fifth International Conference on Distributed Artificial Intelligence*, pp. 1–9, 2023.

[65] Rui Zhao, Jinming Song, Yufeng Yuan, Haifeng Hu, Yang Gao, Yi Wu, Zhongqian Sun, and Wei Yang. Maximum entropy population-based training for zero-shot human-ai coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pp. 6145–6153, 2023.

[66] Luisa Zintgraf, Sam Devlin, Kamil Ciosek, Shimon Whiteson, and Katja Hofmann. Deep interactive bayesian reinforcement learning via meta-learning. *arXiv preprint arXiv:2101.03864*, 2021.

Supplementary Materials

The following content was not necessarily subject to peer review.

Appendix

Please find an anonymized version of the code for this paper at https://anonymous.4open.science/r/rotate/.

A Related Work (Full)

Agent Training in AHT & ZSC. The training of adaptive ego agent policies that can near-optimally collaborate with diverse previously unseen teammates has been explored in AHT [52] & ZSC [21]. Given access to a set of training teammate policies, AHT methods [34] train ego agents to model teammates [3] by identifying important characteristics of different teammates based on their observed behavior. These methods then train a model estimating the best-response policy to the encountered teammate policies based on their inferred characteristics. Recent AHT methods [42, 40, 66, 20, 57] typically use neural networks trained using reinforcement learning [50, 35]. To further improve AHT training, several approaches learn a distribution for sampling teammate policies during training based on maximizing the worst-case returns [56] or regret [17, 11] of trained agents, while other approaches seek to improve the ability of an AHT agent to adapt to unseen teammates at deployment time [37, 38]. As an alternative to AHT, ZSC designs learning methods promoting near-optimal collaboration between agents that have not interacted with each other as long as they learn using the same ZSC algorithm. ZSC methods [21, 22, 13] typically achieve this goal by encouraging the agents to converge towards the same equilibrium despite being trained independently. These agent training algorithms are a crucial component of ROTATE, an iterative training procedure that uses AHT training algorithms to improve an AHT policy based on an interaction with generated teammate policies.

Teammate Generation for AHT & ZSC. Recent work improves existing AHT and ZSC agent training algorithms by designing a diverse collection of training teammate policies. FCP [53] generates the pretrained teammate policies by running the same CMARL algorithm across different seeds. Other work improved FCP by optimizing information-theoretic diversity metrics based on Jensen-Shannon divergence [33], mutual information [32], or entropy [61, 65], which encourages each teammate to yield different trajectories or policies. Recent methods [43, 9, 14, 44, 48] enable the trained AHT agent to learn distinct strategies during training by generating teammate policies requiring distinct best response policies through the maximization of adversarial diversity metrics, which strongly resembles ROTATE's notion of cooperative regret. However, instead of maximizing the regret of the trained AHT agent like ROTATE, these methods maximize the regret from using a generated teammate policy to interact with another generated teammate policy. Unlike ROTATE's open-ended training process, these methods also only generate a limited and fixed set of teammate policies once before agent training. Notably, these methods maximize regret only on the initial state of a collaboration trajectory, leading to sabotaging teammate policies [14, 48] that execute detrimental actions for cooperation in states that it will not visit in self-play. Learning to collaborate with sabotaging teammates is difficult, leading to the proposal of heuristics to reduce sabotage in previous work [14, 48], and a more systematic objective in ROTATE.

Open-Ended Learning (OEL). Our proposed method is heavily influenced by prior work in OEL [28, 54], which explores algorithms that continually design novel tasks to create a generally capable agent [23, 4]. Many OEL methods in RL [58, 15, 41, 25, 24, 46] focus on the problem of unsupervised environment design (UED), which aims to improve RL agent generalization across different tasks by designing and sampling novel environments with different transition and reward functions. Similar to our method, PAIRED [15] trains a neural network using reinforcement learning to output novel environment parameters that induce high regret to the trained ego agent. Other methods assume access to a procedural environment generator, and focus on designing task curators that sample training environments based on criteria such as the expected returns of different policies [58], TD-Error induced during learning [25], regret [24], or learnability [46]. In the context of competitive multi-agent RL, OEL methods seek to generate new opponents for competitive gameplay, often through self-play [51, 29]. For AHT, Yuan et al. [64] and Yan et al. [62] also proposed

open-ended methods that keep generating novel teammate policies for an AHT agent to learn from. Unlike ROTATE, their approach to generating teammates either relies on evolutionary methods to generate new teammates or uses random perturbations of the AHT agent's policy as the new teammate, making it less efficient at producing representative samples from the vast teammate policy set.

B Algorithms

Algorithm 1 Open-Ended Ad Hoc Teamwork

Require:
Environment, Env.
Total of training iterations, T^{iter}.
Initial ego agent policy parameters, θ^{ego}.

1: $\text{B}_\pi \leftarrow \langle\rangle$ ▷ Init teammate policy parameter buffer.
2: **for** $j = 1, 2, \ldots, T^{\text{iter}}$ **do**
3: $\quad \text{B}_\pi^{\text{new}} \leftarrow$ **TeammateGenerator**(Env, θ^{ego}, B_π)
4: $\quad \theta^{\text{ego}} \leftarrow$ **EgoUpdate**(Env, θ^{ego}, $\text{B}_\pi^{\text{new}}$)
5: $\quad \text{B}_\pi \leftarrow \text{B}_\pi^{\text{new}}$
6: **end for**
7: **Return** θ^{ego}

B.1 Framework for Open Ended Ad Hoc Teamwork

Section 5 described an open-ended training framework for training an ego agent that can effectively collaborate with previously unseen teammates. We further detail this general open-ended framework in Algorithm 1. In Line 3, a **TeammateGenerator** function determines a buffer of teammate policy parameters, $\text{B}_\pi^{\text{new}}$. The teammate generator function considers the ego agent's current policy parameters, θ^{ego}, and the previous buffer of teammate policy parameters, B^{new}. Ideally, the teammate generation function generates and samples teammates that induce learning challenges to π^{ego}. In Line 4, an **EgoUpdate** function specifies a procedure that updates the ego agent's policy parameters based on the $\text{B}_\pi^{\text{new}}$ designed by the teammate generator. Pseudocode for ROTATE, which follows the open-ended framework specified by Algorithm 1, is presented in the following section.

B.2 ROTATE Algorithm

ROTATE's teammate generation algorithm is detailed in Algorithm 2. As described in Section 6.1, this teammate generation algorithm jointly trains the parameters of a teammate policy and an estimate of its best response (BR) policy, based on a provided ego agent policy. The parameters of the teammate and BR policies, θ^{-i} and θ^{BR}, are initialized in Line 1. The parameters of the BR critic network, σ^{BR}, are initialized in Line 2, while those for the teammate, $\sigma^{-i,\text{BR}}$ and $\sigma^{-i,\text{ego}}$, are initialized in Line 3. Note that the teammate maintains two critics, for separately estimating returns when interacting with the BR and ego agent policies.

The training of the teammate and BR policies is based on the SP, XP, and SXP interaction data gathered in Lines 5 to 7, which we previously motivated and described in Section 6.1. Recall that an SXP interaction require resetting an environment to start from an available XP state. Since resetting from all available XP states for SXP interaction is impractical, ROTATE *samples* from XP states to obtain start states for SXP interactions. Experiences from SP, XP, and SXP interaction are stored in buffers $D_{\text{SP}}, D_{\text{XP}}, D_{\text{SXP}}$ in the form of a collection of tuples, $D = \langle (s_k, a_k, r_k, s'_k) \rangle_{k=1}^{|D|}$. Lines 11 to 21 of Algorithm 2 then highlight how we use the stored experiences to compute loss functions that the trained models optimize.

Algorithm 2 ROTATE TeammateGenerator Function

Require:

Environment, Env.

Ego agent policy, $\pi_{\theta^{\text{ego}}}$.

Current teammate policy parameter buffer, B_π.

Number of updates, N_{updates}.

PPO clipping parameter, ϵ.

PPO update epochs, N_{epochs}.

$\theta^{-i}, \theta^{\text{BR}} \leftarrow$ **RandomInit**(π), **RandomInit**(π)

$\sigma^{\text{BR}} \leftarrow$ **RandomInit**(V)

$\sigma^{-i,\text{BR}}, \sigma^{-i,\text{ego}} \leftarrow$ **RandomInit**(V), **RandomInit**(V) ▷ Init teammate and BR parameters

for $t_{\text{update}} = 1, 2, \ldots, N_{\text{updates}}$ **do**

  $D_{\text{SP}}, D_{\text{XP}} \leftarrow$ **Interact**$(\pi_{\theta^{\text{BR}}}, \pi_{\theta^{-i}}, p_0^{\text{Env}})$, **Interact**$(\pi_{\theta^{\text{ego}}}, \pi_{\theta^{-i}}, p_0^{\text{Env}})$

  $s_{\text{XP}} \leftarrow$ **SampleStates**(D_{XP}) ▷ Sample XP states

  $D_{\text{SXP}} \leftarrow$ **Interact**$(\pi_{\theta^{\text{BR}}}, \pi_{\theta^{-i}}, \mathcal{U}(s_{\text{XP}}))$ ▷ Gather SP, XP, and SXP data

  $\theta^{\text{BR}}_{\text{old}}, \theta^{-i}_{\text{old}} \leftarrow \theta^{\text{BR}}, \theta^{-i}$

  $\sigma^{\text{BR}}_{\text{old}}, \sigma^{-i,\text{BR}}_{\text{old}}, \sigma^{-i,\text{ego}}_{\text{old}} \leftarrow \sigma^{\text{BR}}, \sigma^{-i,\text{BR}}, \sigma^{-i,\text{ego}}$ ▷ Store old model parameters.

  for $k_{\text{update}} = 1, 2, \ldots, N_{\text{epochs}}$ **do**

    $L_{\text{ppo-clip}}(\theta^{\text{BR}}) \leftarrow$ **POL_LOSS_ADV_TARG** $\left(\theta^{\text{BR}}\theta^{\text{BR}}_{\text{old}}, \sigma^{\text{BR}}_{\text{old}}, D_{\text{SP}} \cup D_{\text{SXP}}, \epsilon\right)$

    $L_{\text{ppo-clip}}(\theta^{-i}) \leftarrow$ **POL_LOSS_ADV_TARG** $\left(\theta^{-i}, \theta^{-i}_{\text{old}}, \sigma^{-i,\text{BR}}_{\text{old}}, D_{\text{SP}} \cup D_{\text{SXP}}, \epsilon\right)$

    $L_{\text{reg}}(\theta^{-i}) \leftarrow$ **POL_LOSS_REG_TARG** $\left(\theta^{-i}, \theta^{-i}_{\text{old}}, \sigma^{-i,\text{BR}}_{\text{old}}, \sigma^{-i,\text{ego}}_{\text{old}}, D_{\text{XP}}, \epsilon\right)$

    $L_V(\sigma^{\text{BR}}) \leftarrow$ **VAL_LOSS**$(\sigma^{\text{BR}}, \sigma^{\text{BR}}_{\text{old}}, D_{\text{SP}} \cup D_{\text{SXP}})$

    $L_V(\sigma^{-i,\text{BR}}) \leftarrow$ **VAL_LOSS** $\left(\sigma^{-i,\text{BR}}, \sigma^{-i,\text{BR}}_{\text{old}}, D_{\text{SP}} \cup D_{\text{SXP}}\right)$

    $L_V(\sigma^{-i,\text{ego}}) \leftarrow$ **VAL_LOSS** $\left(\sigma^{-i,\text{ego}}, \sigma^{-i,\text{ego}}_{\text{old}}, D_{\text{XP}}\right)$

    $\theta^{\text{BR}} \leftarrow$ **GradDesc**$(\theta^{\text{BR}}, \nabla_{\theta^{\text{BR}}} L_{\text{ppo-clip}}(\theta^{\text{BR}}))$

    $\theta^{-i} \leftarrow$ **GradDesc** $\left(\theta^{-i}, \nabla_{\theta^{-i}} \left(L_{\text{ppo-clip}}(\theta^{-i}) + L_{\text{reg}}(\theta^{-i})\right)\right)$ ▷ Update policies

    $\sigma^{\text{BR}} \leftarrow$ **GradDesc**$(\sigma^{\text{BR}}, \nabla_{\sigma^{\text{BR}}} L_V(\sigma^{\text{BR}}))$

    $\sigma^{-i,\text{BR}} \leftarrow$ **GradDesc**$(\sigma^{-i,\text{BR}}, \nabla_{\sigma^{-i,\text{BR}}} L_V(\sigma^{-i,\text{BR}}))$

    $\sigma^{-i,\text{ego}} \leftarrow$ **GradDesc**$(\sigma^{-i,\text{ego}}, \nabla_{\sigma^{-i,\text{ego}}} L_V(\sigma^{-i,\text{ego}}))$ ▷ Update critics.

  end for

end for

$\mathrm{B}_\pi \leftarrow \mathrm{B}_\pi \cup \langle \theta^{-i} \rangle$ ▷ Add generated teammate policy parameter

Return B_π

Lines 11 and 12 describe how the teammate and BR policies are trained to mutually maximize returns when interacting with each other during SP and SXP interactions. Both lines call the **POL_LOSS_ADV_TARG** function, which receives $(\theta, \theta_{\text{old}}, \sigma_{\text{old}}, D, \epsilon)$ as input to evaluate the following, standard PPO-clip loss function that encourages return maximization and sufficient exploration:

$$\mathop{\mathbb{E}}_{(s,a,r,s')\in D}\left[\underbrace{-\min\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}A, \text{clip}\left(\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}, 1-\epsilon, 1+\epsilon\right)A\right)}_{\text{PPO Clip Loss}} + \underbrace{\pi_\theta(a|s)\log\left(\pi_\theta(a|s)\right)}_{\text{Entropy Loss}}\right],$$

where A denotes the *advantage* function. Our implementation of ROTATE uses an estimate of the advantage function obtained via the Generalized Advantage Estimation (GAE) algorithm [49], $A^{\text{GAE}}_{\sigma_{\text{old}}}$. Meanwhile, Line 13 shows how the teammate policy is trained to maximize the ego agent's *regret* based on experiences from XP interaction. The **POL_LOSS_REG_TARG** function that computes a loss function that encourages the maximization of regret is generally the same as the **POL_LOSS_ADV_TARG** function except for its replacement of the advantage function, A, with a regret-based target function defined below:

$$A_{\text{reg}} = \underbrace{V_{\sigma_{\text{old}}^{-i,\text{BR}}}(s)}_{\approx V(s|\pi^{-i},\text{BR}(\pi^{-i}))} - \underbrace{\left(r + \gamma V_{\sigma_{\text{old}}^{-i,\text{ego}}}(s')\right)}_{\approx V(s|\pi^{-i},\pi^{\text{ego}})}. \tag{11}$$

Rather than optimizing a regret function that requires explicitly computing the return-to-go for SP and XP interaction starting from state s, **POL_LOSS_REG_TARG** *estimates* the XP return via a 1-step bootstrapped return using the teammate critic parameterized by $\sigma^{-i,\text{BR}}$. Similarly, the SP return is estimated using the teammate critic network parameterized by $\sigma^{-i,\text{ego}}$. This results in a regret optimization method that uses the log-derivative trick to optimize objective functions [60, 19]. The ROTATE regret estimation method and alternative approaches to maximize regret are further discussed in App. E.2.

Lines 14 to 16 then detail how we train critic networks that measure returns from the interaction between the generated teammate policy and its best response or ego agent policy. We specifically call the **VAL_LOSS** function that receives $(\sigma, \sigma_{\text{old}}, D)$ to compute the standard mean squared Bellman error (MSBE) loss, defined as:

$$\mathop{\mathbb{E}}_{(s,a,r',s')\in D}\left[\left(V_\sigma(s) - V^{\text{targ}}_{\sigma_{\text{old}}}(s)\right)^2\right], \tag{12}$$

where $V^{\text{targ}}_{\sigma_{\text{old}}}(s) := A^{\text{GAE}}_{\sigma_{\text{old}}} - V_{\sigma_{\text{old}}}(s)$ is the target value estimate.

The previously defined loss functions can be minimized using any gradient descent-based optimization technique, as we indicate in Lines 17 to 21. In practice, our implementation uses the ADAM optimization technique [26]. At the end of this teammate generation process, Lines 24 and 25 indicate how the generated teammate policy parameter is added to a storage buffer, which is subsequently uniformly sampled to provide teammate policies for ego agent training.

The ego agent policy's training process proceeds according to Algorithm 3. Line 3 illustrates how ROTATE creates different teammate policies by uniformly sampling model parameters from the B_π resulting from the teammate generation process. Using the experience collaborating with the sampled policies outlined in Line 4, the ego agent's policy parameters are updated to maximize its returns via PPO in Line 7. The only difference between the **EGO_POL_LOSS** function and **POL_LOSS_ADV_TARG** function in Algorithm 2 is the input used to compute the loss function. Unlike in the **EGO_POL_LOSS** function, we assume that the input dataset, D, stores the historical sequence of observed states and executed actions, h, rather than states. Likewise, we assume that the only difference between the **VAL_LOSS** and **EGO_VAL_LOSS** function is that the latter stores the observation-action history rather than states (Line 8). Like recent AHT learning algorithms [66, 42, 40], π^{ego} and V^{ego} are conditioned on the ego agent's observation-action history to facilitate an

Algorithm 3 ROTATE EgoUpdate Function

Require:
Environment, Env.
Ego agent policy parameters, θ^{ego}.
Current teammate policy parameter buffer, B_π.
Number of updates, N_{updates}.
PPO clipping parameter, ϵ.
PPO update epochs, N_{epochs}

$\sigma^{\text{ego}} \leftarrow$ **Init**(V) ▷ Init params of the critic networks of π^{ego}
for $t_{\text{update}} = 1, 2, \ldots, N_{\text{updates}}$ **do**
  $\theta^{-i} \sim \mathcal{U}(\text{B}_\pi)$ ▷ Sample teammate parameters uniformly
  $D \leftarrow$ **Interact**$(\pi_{\theta^{-i}}, \pi_{\theta^{\text{ego}}}, p_0^{\text{Env}})$
  $\theta^{\text{ego}}_{\text{old}}, \sigma^{\text{ego}}_{\text{old}} \leftarrow \theta^{\text{ego}}, \sigma^{\text{ego}}$
  for $k_{\text{update}} \in \{1, 2, \ldots, N_{\text{epochs}}\}$ **do**
    $L_\pi(\theta^{\text{ego}}) \leftarrow$ **EGO_POL_LOSS** $\left(\theta^{\text{ego}}, \theta^{\text{ego}}_{\text{old}}, \sigma^{\text{ego}}_{\text{old}}, D, \epsilon\right)$ ▷ Compute policy loss
    $L_V(\sigma^{\text{ego}}) \leftarrow$ **EGO_VAL_LOSS** $\left(\sigma^{\text{ego}}, \sigma^{\text{ego}}_{\text{old}}, D, \epsilon\right)$ ▷ Compute critic loss
    $\theta^{\text{ego}} \leftarrow$ **GradDesc**$(\theta^{\text{ego}}, \nabla_{\theta^{\text{ego}}} L_\pi(\theta^{\text{ego}}))$ ▷ Update policy
    $\sigma^{\text{ego}} \leftarrow$ **GradDesc**$(\sigma^{\text{ego}}, \nabla_{\sigma^{\text{ego}}} L_V(\sigma^{\text{ego}}))$ ▷ Update critic
  end for
end for
Return θ^{ego}

adaptive π^{ego} through an improved characterization of teammates' policies. The history-conditioned ego architecture and other practical implementation details are described in App. G. Finally, the ego agent update function returns the updated ego agent policy parameters, which are provided as part of the inputs for the next call to ROTATE's teammate generation function.

C Baselines Overview

The main paper compares ROTATE to five baselines: PAIRED, Minimax Return, FCP, BRDiv, and CoMeDi. Each baseline is briefly described below, followed by a discussion of the computational complexity of teammate generation baselines compared to ROTATE, and a discussion of the relationship of Mixed Play (MP) with per-state and per-trajectory regret. A discussion of implementation details can be found in App. G.

PAIRED [15]: A UED algorithm where a regret-maximizing"adversary" agent proposes environment variations that an allied antagonist achieves high returns on, but a protagonist agent receives low returns on. The algorithm is directly applicable to AHT by defining a teammate generator for the role of the adversary, a best response agent to the generated teammate for the role of the antagonist, and an ego agent for the role of the protagonist.

Minimax Return [36, 56]: A common baseline in the UED literature, with origins in robust reinforcement learning, where the objective is minimax return. Prior works in AHT have proposed generating a curriculum of teammates according to this objective. Translated to our open-ended learning setting, the teammate generator creates teammates that minimize the ego agent's return, while the ego agent maximizes return.

Fictitious Co-Play [53]: A two-stage AHT algorithm where a pool of teammates is generated by running IPPO [63] with varying seeds, and saving multiple checkpoints to the pool. The ego agent is an IPPO agent that is trained against the pool.

BRDiv [43]: A two-stage AHT algorithm where a population of "confederate" and best-response agent pairs is generated, and an ego agent is trained against the confederates. BRDiv maintains a cross-play matrix containing the returns for all confederate and best-response pairs. The diagonal returns (self-play) are maximized, while the off-diagonal returns (cross-play) are minimized. BRDiv and LIPO [9] share a similar objective, where the main differences are: (1) If `xp_weight` denotes the weight on the XP return, then BRDiv requires that the coefficient on the SP return is always $1 + 2*$`xp_weight`, and (2) LIPO introduces a secondary diversity metric based on mutual information, and (3) LIPO assumes that agents within a team (i.e., a confederate-BR pair) share parameters.

CoMeDi [48]: CoMeDi is a two-stage AHT algorithm. In the first stage, a population of teammates is generated, and in the second stage, an ego agent is trained against the teammate population. The teammate generation stage trains teammate policies one at a time, where the nth teammate policy is trained to maximize its SP return, minimize its XP return with the previously generated teammate (i.e. from among teammates $1, \cdots, n-1$) that it best collaborates with, and maximizes its "mixed-play" (MP) return. The relationship between the regret objectives described in Section 6 and MP is further discussed in App. E.1.

C.1 Computational Complexity of ROTATE versus Teammate Generation Baselines

The computational complexity of ROTATE is compared with that of the teammate generation baselines, in terms of the population size and the number of objective updates. In the following, n denotes the population size, while T indicates the number of updates needed to train an individual population member. The precise meaning of n and T might vary with the algorithm, but is made clear in each description.

FCP: Let T denote the number of RL updates needed to train each IPPO team and let n denote the number of teams trained by FCP. Then, the computational complexity of FCP is $O(nT)$.

BRDiV/LIPO: Both BRDiv [43] and LIPO [9] require sampling trajectories from each pair of agents in the population, for each update. Thus, if the total number of updates is T and the population size is n, then the algorithm's time complexity is $O(n^2T)$. Due to the quadratic complexity in n, BRDiv and LIPO are typically run with smaller population sizes, with $n < 10$ for all non-matrix game tasks in both original papers.

CoMeDi: Recall that CoMeDi trains population members one at a time, such that each agent is distinct from the previously discovered teammates in the population. This necessitates performing evaluation rollouts of the currently trained agent against all previously generated teammates at each RL update step. Let T be the number of RL updates required to train the ith agent to convergence, and let n denote the population size. Then CoMeDi's time complexity is $O(n^2T)$—making it scale quadratically in n, similar to BRDiv and LIPO.

ROTATE: In ROTATE, a new teammate is trained to convergence for each iteration of open-ended learning. Thus, the number of open-ended learning iterations is equal to the population size n, where within each iteration, there are $O(T)$ RL updates performed. Therefore, the complexity of ROTATE is $O(nT)$, meaning that our method scales linearly in the population size n.

D Experimental Results: RQ3 and RQ4

This section presents the results and analysis for RQ3 and RQ4 from Section 7.

RQ3: Is the population buffer necessary for ROTATE to learn well? (Yes) We hypothesize that collecting all previously generated teammates in a population buffer helps the ROTATE agent improve in robustness against all previously discovered conventions. On the other hand, if there is no

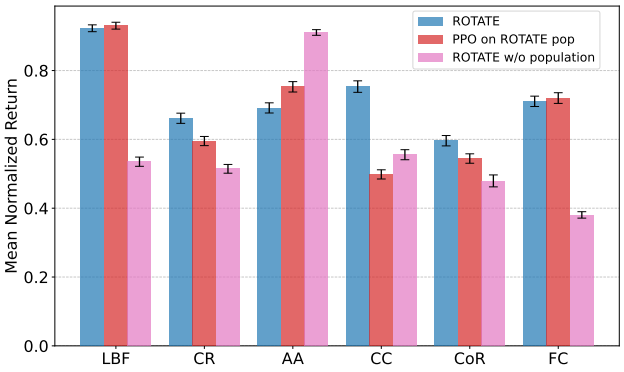


Figure 4: ROTATE compared to an independently trained ego agent on ROTATE's population, and an ablation of ROTATE without the population. The mean normalized return and 95% bootstrapped CI's are shown.

population buffer, then it becomes possible for the ROTATE ego agent to forget how to collaborate with teammate seen at earlier iterations of open-ended learning [27], which creates the possibility that the ego agent and teammate generator oscillates between conventions. As shown in Fig. 6, ROTATE without the population buffer attains lower evaluation returns than the full ROTATE method on all tasks except for AA, thus supporting the hypothesis that the population buffer improves ego agent learning. As discussed in **RQ1**, AA is a unique layout where agents can complete the task independently, even in the presence of an adversarial partner. As a corollary, there are few meaningful cooperative conventions that can be discovered, and no scenarios where convention mismatch leads to zero return (unlike LBF and FC).

RQ4: Is the population generated by ROTATE useful for training an independent ego agent? (Yes) Two-stage AHT algorithms first generate a population of teammates, and next train an ego agent against the population. Although ROTATE's teammate generation mechanism relies on the learning process of a particular ego agent, here, we investigate whether the population generated by ROTATE is useful for training independently generated ego agents. Fig. 4 (presented in the Appendix) compares the mean evaluation returns of the ROTATE ego agent against the mean evaluation returns of an independently trained ego agent that was trained using the same configuration as ROTATE. In 3/6 tasks, the ROTATE ego agent outperforms the independently trained ego agent, while in two tasks, the two ego agents perform similarly (LBF and FC). Thus, the experiment suggests that the ROTATE population is a useful population of teammates even independent of the particular ego agent generated. The strong performance of the independently trained ego agent is unsurprising given that it has two advantages over the ROTATE ego agent. First, the independently trained ego agent faces a stationary distribution of training teammates compared to ROTATE, which faces a nonstationary distribution caused by the population growing over learning iterations. Second, the independently trained ego agent interacts with all teammates uniformly throughout training, while the ROTATE ego agent only trains against earlier teammates for more iterations than later teammates.

E Supplemental Results

This section presents various supplemental results. First, we describe CoMeDi's mixed-play mechanism in the context of ROTATE's per-state regret. Second, we discuss alternative estimators for ROTATE per-state regret. Third, we present experiments comparing ROTATE to a variant with CoMeDi-style mixed-play return maximization, and a variant using the alternative regret estimation strategy. Next, we present and describe radar charts breaking down the performance of ROTATE

on all six tasks presented in the main paper. Finally, we present the learning curves for all variants of ROTATE that are tested in this paper.

E.1 Discussion of CoMeDi and Mixed Play

As previously described in App. C, CoMeDi [48] is a two-stage teammate generation AHT algorithm, whose teammate generation process trains one teammate per iteration, with an objective that encourages the new teammates to be distinct from previously discovered teammates.

CoMeDi adds trained teammates policies to a teammate policy buffer, Π^{train}. Each iteration begins by identifying the teammate policy that is most *compatible* with the currently trained teammate π^{-i}, out of all previously generated policies:

$$\pi^{\text{comp}} = \underset{\pi^{-j} \in \Pi^{\text{train}}}{\text{argmax}} \, \mathbb{E}_{s \sim p_0}[V(s|\pi^{-i}, \pi^{-j})]. \tag{13}$$

The new teammate policy π^{-i} is trained with an objective that improves the per-trajectory regret objective (Eq. 9) by adding a term that maximizes the returns from states gathered in *mixed-play*, which we describe below.

Let mixed-play starting states be sampled from states visited when π^{-i} interacts with the *mixed* policy, that uniformly samples actions from π^{comp} and $\text{BR}(\pi^{-i})$ at each timestep:

$$p_{\text{MSTART}} := d\left(\pi^{-i}, \frac{1}{2}\pi^{\text{comp}} + \frac{1}{2}\text{BR}(\pi^{-i}); p_0\right). \tag{14}$$

From these starting states, CoMeDi then gathers mixed-play interaction data, where π^{-i} interacts with $\text{BR}(\pi^{-i})$. The resulting mixed-play state visitation is then expressed as:

$$p_{\text{MP}} := d\left(\pi^{-i}, \text{BR}(\pi^{-i}); p_{\text{MSTART}}\right). \tag{15}$$

The complete objective that Sarkar et al. [48] optimizes to train a collection of diverse teammates is then defined as:

$$\max_{\pi} \left(\mathbb{E}_{s_0 \sim p_0}\left[\text{CR}(\pi^{\text{comp}}, \pi^{-i}, s_0)\right] + \underbrace{\mathbb{E}_{s \sim p_{\text{MP}}}[V(s|\pi, \text{BR}(\pi))]}_{\text{mixed-play return maximization}}\right). \tag{16}$$

CoMeDi [48] optimizes this objective to discourage π^{-i} from learning poor actions for collaborations outside of p_{SP}. This is because π^{-i} is now also trained to maximize returns in states visited during mixed-play, which resembles some states encountered while cooperating with π^{comp}. Discerning whether a state is likely encountered while interacting with π^{comp} and consequently choosing to sabotage collaboration will no longer be an optimal policy to maximize Expr. 16.

Despite the importance of using p_{MSTART} as a starting state for data collection being questionable, we take inspiration from CoMeDi's maximization of $V(s|\pi, \text{BR}(\pi))$ outside of states from p_{SP}. We argue that maximizing $V(s|\pi^{-i}, \text{BR}(\pi^{-i}))$ is a key component towards making π^{-i} act in good faith by always choosing actions yielding optimal collective returns assuming $\text{BR}(\pi^{-i})$ is substituted as the partner policy. Unlike CoMeDi, ROTATE maximizes $V(s|\pi^{-i}, \text{BR}(\pi^{-i}))$ on trajectories gathered from a starting state from p_{XP} instead of p_{MSTART}, which results in the second term of Expr. 10. We formulate this objective to encourage π^{-i} to act in good faith in states sampled from p_{XP}, which is visited while π^{-i} interacts with π^{ego}. Since π^{-i} is not sabotaging π^{ego} by selecting actions that make collaboration impossible in p_{XP}, the ego policy learning process becomes less challenging. We conjecture that this leads to π^{ego} with better performances as indicated in Figure 3.

While Figure 3 compares ROTATE with CoMeDi, Figure 6 compares ROTATE with a modified CoMeDi approach that now follows the open-ended training framework described in Algorithm 1. In this modified version of CoMeDi, we train a newly generated teammate policy to maximize Eq. 16

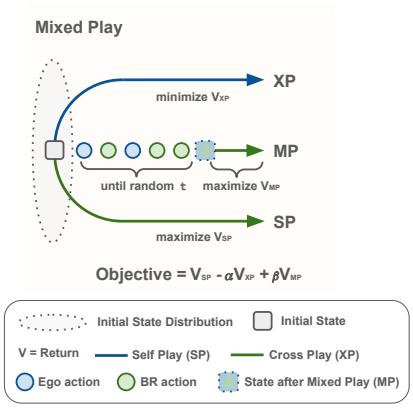


Figure 5: CoMeDi-style mixed-play objective for teammate generation, in the context of open-ended AHT.

while substituting π^{comp} with the trained π^{ego}. Rather than promoting meaningful differences with previously generated teammate policies, this creates a teammate policy that maximizes the ego agent policy's per-trajectory regret while mitigating self-sabotage. This version of CoMeDi's teammate generation objective within the ROTATE open-ended framework is visualized in Figure 5.

E.2 Alternatives Estimators for Per-State Regret

This section discusses the approach employed by ROTATE in Algorithm 2 to estimate the per-state regret objective under a specific distribution, as well as an alternative estimation method. Experiments comparing the two approaches are also presented and discussed.

Recall that the per-state regret under states sampled from a distribution D is defined as:

$$\mathbb{E}_{s\sim D}[\text{CR}(\pi^{\text{ego}}, \pi^{-i}, s)] = \mathbb{E}_{s\sim D}\left[V\left(s|\pi^{-i}, \text{BR}(\pi^{-i})\right) - V\left(s|\pi^{-i}, \pi^{\text{ego}}\right)\right] \tag{17}$$

$$= \underbrace{\mathbb{E}_{s\sim D}\left[V\left(s|\pi^{-i}, \text{BR}(\pi^{-i})\right)\right]}_{\text{SP return}} - \underbrace{\mathbb{E}_{s\sim D}\left[V\left(s|\pi^{-i}, \pi^{\text{ego}}\right)\right]}_{\text{XP return}}. \tag{18}$$

In practice, we can use the policy gradient method to maximize regret by estimating the self-play returns and cross-play returns in Eq. 18 using the n-step return, Monte Carlo-based return-to-go estimate, or generally any variant of the advantage function estimator. The choice of return estimates affects the result of our teammate generation process through the bias-variance tradeoff when estimating regret. Combined with the potentially different choices of D, we can design different variants of ROTATE based on how regret is estimated.

ROTATE Per-State Regret: Line 13 in Algorithm 2 and Eq. 11 outline how ROTATE maximizes per-state regret in states visited during XP interaction (denoted by p_{XP}), where SP and XP returns are estimated via a trained critic and a 1-step return estimate, respectively. As a reminder, ROTATE employs the following target function to train the regret-maximizing teammate policy:

$$\mathbb{E}_{s\sim p_{\text{XP}}}\left[\underbrace{V_{\sigma^{-i},\text{BR}}(s)}_{\text{SP return estimate}} - \underbrace{\left(r + \gamma V_{\sigma^{-i},\text{ego}}(s')\right)}_{\text{XP return estimate}}\right]. \tag{19}$$

We maximize regret in states sampled from p_{XP} to encourage the design of teammate policies that provide a learning challenge while also acting in good faith, thereby maximizing cooperative returns assuming interactions with its best-response policy, while interacting with the ego agent's policy. Despite potentially providing biased estimates, training a value function to estimate self-play returns can reduce the variance caused by environment stochasticity, compared to a Monte Carlo return-to-go estimate.

The critic network estimating teammate-BR returns, $V_{\sigma^{-i,\text{BR}}}(s)$, is trained on interactions initialized from states sampled from p_{XP} (SXP interactions), as shown in Line 15 of Algorithm 2. This enables the teammate-BR critic network to accurately estimate SP returns from p_{XP} states. Meanwhile, a 1-step estimate of XP returns is made possible by storage of rewards experienced during XP interactions (Line 5 of Algorithm 2) and the training of a value function to estimate XP returns (Line 16 of Algorithm 2). Utilizing a 1-step estimate produces lower variance than using a Monte Carlo-based return-to-go estimate, while also yielding less bias than predicting returns solely based on the trained critic network's value.

Estimating Per-State Regret via Monte Carlo Returns: An alternative approach for estimating is to use a Monte Carlo-based return-to-go estimate for both SP and XP return estimates. Assuming that both interaction starts from states encountered during XP interaction, the policy updates under this alternative approach maximize the following target function:

$$\mathbb{E}_{s_t\sim p_{\text{XP}}}\left[\underbrace{\mathbb{E}_{a_{t'}\sim[\text{BR}(\pi^{-i}),\pi^{-i}],P}\left[\sum_{t'=t}^{\infty}\gamma^{t'}r_{t'}\middle|s_t\right]}_{\text{SP return estimate}}-\underbrace{\mathbb{E}_{a_{t'}\sim[\pi^{\text{ego}},\pi^{-i}],P}\left[\sum_{l=0}^{\infty}\gamma^{t'}r_{t'}\middle|s_t\right]}_{\text{XP return estimate}}\right]. \tag{20}$$

We refer to this as the *Monte Carlo* per-state regret. However, starting both SP and XP interactions from all states visited in XP can be computationally prohibitive. More importantly, the Monte Carlo-based return-to-go estimates of SP and XP returns have high variance, especially when the environment transition function and the trained policies are highly stochastic.

Estimating Per-State Regret via Generalized Advantage Estimators: A final approach for estimating Eq. 17 is to substitute both return-to-go estimates in Expr. 20 with a generalized advantage estimator [49] based on SP and XP interactions. This results in the maximization of the following target function during the teammate policy updates:

$$\mathbb{E}_{s_t\sim p_{\text{XP}}}\left[\underbrace{\mathbb{E}_{a_{t'}\sim[\text{BR}(\pi^{-i}),\pi^{-i}],P}\left[\underbrace{\sum_{t'=t}^{\infty}(\gamma\lambda)^{t'}\,\delta_{t'}^{-i,\text{BR}}}_{\text{GAE}}\middle|s_0\right]}_{\text{SP return estimate}}-\underbrace{\mathbb{E}_{a_{t'}\sim[\pi^{\text{ego}},\pi^{-i}],P}\left[\underbrace{\sum_{t'=t}^{\infty}(\gamma\lambda)^{t'}\,\delta_{t'}^{-i,\text{ego}}}_{\text{GAE}}\middle|s_0\right]}_{\text{XP return estimate}}\right], \tag{21}$$

where we define $\delta_t^{-i,\text{BR}}$ and $\delta_t^{-i,\text{ego}}$ as:

$$\delta_t^{-i,\text{BR}} = r_t + \gamma V_{\sigma^{-i,\text{BR}}}(s_{t+1}) - V_{\sigma^{-i,\text{BR}}}(s_t),$$
$$\delta_t^{-i,\text{ego}} = r_t + \gamma V_{\sigma^{-i,\text{ego}}}(s_{t+1}) - V_{\sigma^{-i,\text{ego}}}(s_t).$$

We refer to an instance of the ROTATE algorithm that maximizes regret using this target function as ROTATE with *GAE per-state regret*. In practice, we collect data for SP GAE maximization and XP GAE minimization by first independently sampling two collections of states from D_{SXP} and D_{XP} respectively. Next, the states sampled from D_{SXP} are used to maximize the GAE from SXP interactions, while states sampled from D_{XP} are utilized to minimize the GAE from XP interactions. The γ and λ parameters used during the computation of the generalized advantage estimator are mechanisms to regulate the bias and variance of the regret estimation [49], effectively providing a different bias-variance tradeoff compared to the previously mentioned methods.

E.3 Experimental Comparisons of ROTATE Teammate Generation Objectives

Figure 6 compares the version of ROTATE presented in the main paper and Algorithm 2, to ROTATE with GAE per-state regret, and a version of ROTATE where expected returns are maximized in states sampled from p_{MP} rather than p_{SXP}, which resembles the mixed-play objective of

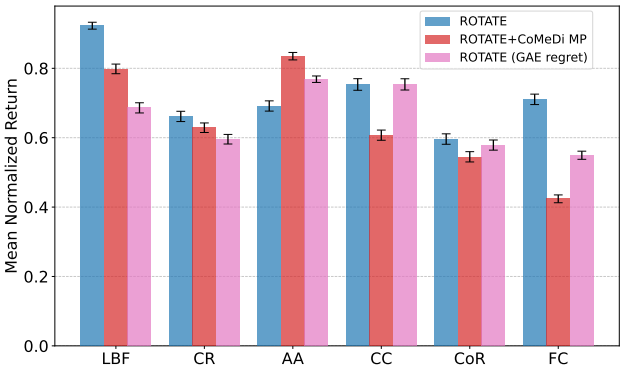


Figure 6: ROTATE vs ROTATE with CoMeDi's mixed-play (MP) objective and ROTATE with GAE regret.

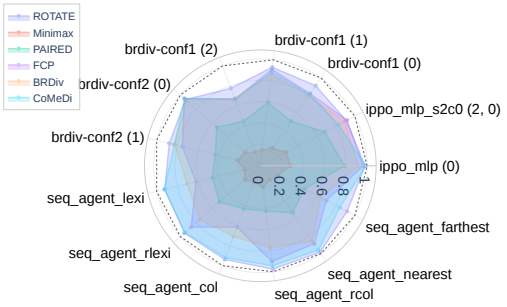
CoMeDi [48]. We do not implement the Monte Carlo per-state regret estimation approach described above, as it is impractical and unlikely to yield better results than using value functions to estimate regret. ROTATE and ROTATE with GAE regret yield mixed results as neither approach consistently beats the other in all environments. We suspect this is caused by the policy gradient's different bias and variance levels when estimating regret using these two methods. Meanwhile, ROTATE's maximization of returns in states from p_{SXP} leads to higher normalized returns than maximizing CoMeDi's mixed-play objective in all environments except for Overcooked's Asymmetric Advantages (AA) setting. Following the difference in starting states of trajectories for which these two maximize self-play returns, we conjecture that this is because ROTATE empirically teammate policies with good faith in states from p_{XP} while the CoMeDi-like approach imposes the same thing in states from p_{MSTART}. Imposing good faith within policies in p_{XP} is likely more important for training an ego agent that initially interacts with π^{-i} during training by visiting states from p_{XP}.

E.4 ROTATE vs Baselines—Radar Charts

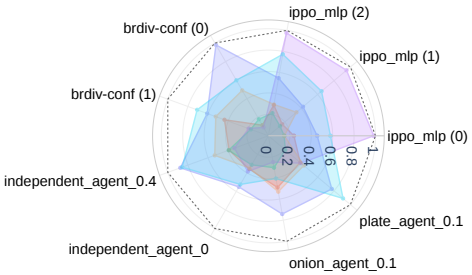
We break down the performance of ROTATE and all baseline methods by individual evaluation teammate policies as radar charts in Fig. 7. The radar charts show that ROTATE achieves higher performance across a larger number and variety of evaluation teammates than baselines. The best baseline, CoMeDi, achieves unusually high returns with the heuristic-based evaluation teammates on LBF, CR, and CC. We hypothesize that this trend occurs because CoMeDi explicitly optimizes for novel conventions that do not match existing conventions. However, on these tasks, CoMeDi does not perform as well as BRDiv teammates, which are trained to maximize the adversarial diversity objective. The radar charts also show that the second-best baseline, FCP, is strong specifically against IPPO teammates and relatively weaker on heuristics and BRDiv teammates, especially in CR and CC. As mentioned in the main paper, we attribute FCP's relative strength on IPPO evaluation teammates to the fact that the IPPO evaluation teammates are closer to the training teammate distribution constructed by FCP. While FCP is not especially strong against the "IPPO pass" agents in CC, these agents were trained via reward shaping to solve the task by passing onions across the counter rather than navigating around the counter, which is the policy found by IPPO without reward shaping (denoted as "IPPO CC" in the figures).

E.5 Learning Curves

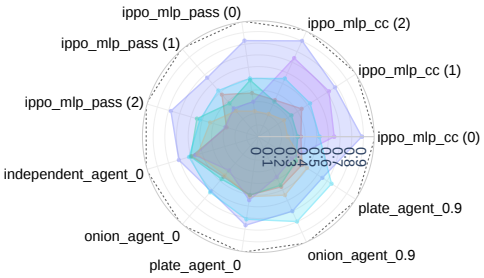
Figure 8 shows learning curves for ROTATE and all ROTATE variations tested in this paper, where the x-axis is the open-ended learning iteration, and the y-axis corresponds to the mean evaluation return. On 4/6 tasks (LBF, CR, CC, and FC), ROTATE has better sample efficiency than variants. On 3/6 tasks (LBF, CR, and FC), ROTATE dominates variants at almost all points in learning.



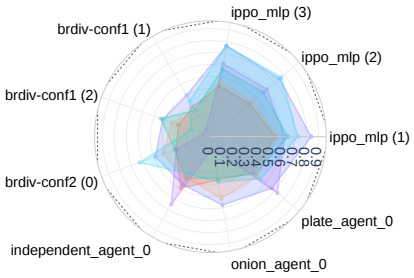
(a) LBF.



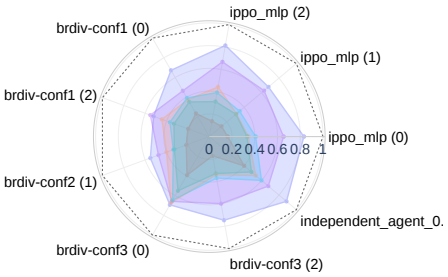
(b) Cramped Room.



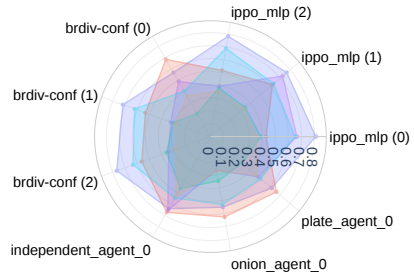
(c) Counter Circuit.



(d) Coordination Ring.



(e) Forced Coordination.



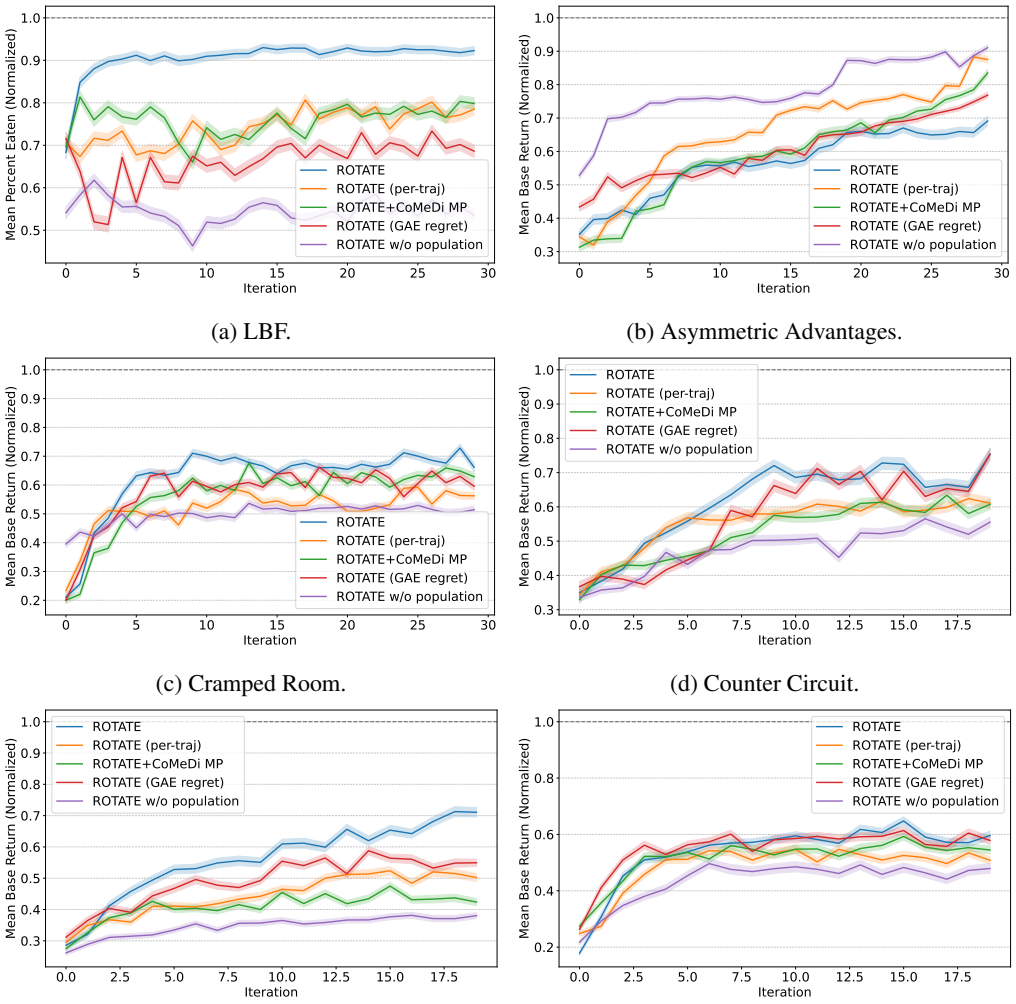
(f) Asymmetric Advantages.

Figure 7: Normalized mean returns of ROTATE and all baselines across all tasks, broken down by evaluation teammate in Π^{eval}. Legend shown for LBF applies for all plots.

F Experimental Tasks

Experiments in the main paper are conducted on Jax re-implementations of Level-Based Foraging (LBF) [2, 5], and five tasks from the Overcooked suite—Cramped Room (CR), Asymmetric Advantages (AA), Counter Circuit (CC), Coordination Ring (CoR), and Forced Coordination (FC) [8, 47]. Each task is described below.

Level-Based Foraging (LBF) Originally introduced by Albrecht & Ramamoorthy [2], Level-Based Foraging is a mixed cooperative-competitive logistics problem where N players interact within a rectangular grid world to obtain k foods. All players and foods have a positive integer



(a) LBF.

(b) Asymmetric Advantages.

(c) Cramped Room.

(d) Counter Circuit.

(e) Forced Coordination.

(f) Coordination Ring.

Figure 8: Learning curves of ROTATE and all variations of ROTATE considered in this paper. Normalized mean returns and bootstrapped 95% confidence intervals on Π^{eval} are shown.

level, where groups of one to four players may only *load* (collect) a food if the sum of player levels is greater than the food's level. A food's level is configured so that it is always possible to load it.

We use the Jax re-implementation of LBF by Bonnet et al. [5], which was based on the implementation by Christianos et al. [12]. The implementation permits the user to specify the number of players, number of foods, grid world size, level of observability, and whether to set the food level equal to the sum to player levels in order to force players to coordinate to load each food.

The experiments in this paper configured the LBF environment to a 7×7 grid, where two players interact to collect three foods. Our LBF configuration is shown in Fig. 9. Each player observes the full environment state, allowing each player to observe the locations of other agents and all foods and the number of time steps elapsed in the current episode. Each player has six discrete actions: up, down, left, right, no-op, and *load*, where the last action is the special food collection action. A food may only be collected if the sum of player levels is greater than the level of the food. Since this paper focuses on fully cooperative scenarios, we set the food level equal to the level of both players, so all foods require cooperation in order to be collected. When a food is collected, both players receive an identical reward, which is normalized such that the maximum return in an episode is 0.5.

An episode terminates if an invalid action is taken, players collide, or when 100 time steps have passed. Player and food locations are randomized for each episode.

Overcooked Introduced by Carroll et al. [8], the Overcooked suite is a set of two-player collaborative cooking tasks, based on the commercially successful Overcooked video game. Designed to study human-AI collaboration, the original Overcooked suite consists of five simple environment *layouts*, where two agents collaborate within a grid world kitchen to cook and deliver onion soups. While Carroll et al. [8] introduced Overcooked to study human-AI coordination, Overcooked has become popularized for AHT research as well [9, 48, 17].

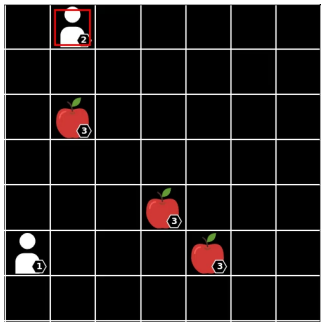
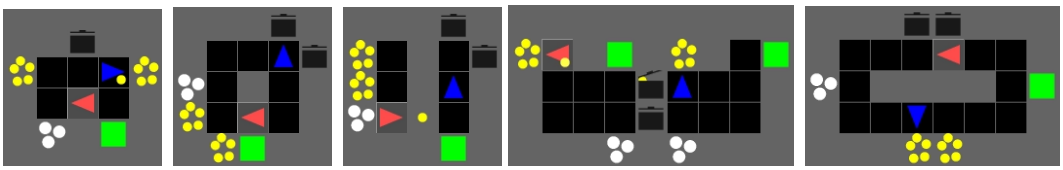


Figure 9: Level-based foraging environment. The apple icons denote food. The number on the icon indicates each player's and food's level. The AHT player is indicated by the red box.

We use the Jax re-implementation of the Overcooked suite by Rutherford et al. [47], which is based on the original implementation by Carroll et al. [8]. Later versions of Overcooked include features such as multiple dish types, order lists, and alternative layouts, but this paper considers only the five original Overcooked layouts: Cramped Room (CR), Asymmetric Advantages (AA), Counter Circuit (CC), Coordination Ring (CoR), and Forced Coordination (FC).

The objective for all five tasks is to deliver as many onion soups as possible, where the only difference between the tasks is the environment layout, as shown in Fig. 10. To deliver an onion soup, players must place three onions in a pot to cook, use a plate to pick up the cooked soup, and send the plated soup to the delivery location. Each player observes the state and location of all environment features (counters, pots, delivery, onions, and plates), the position and orientation of both players, and an urgency indicator, which is 1 if there are 40 or fewer remaining time steps, and 0 otherwise. Each player has six discrete actions, consisting of the four movement actions, interact, and no-op. The reward function awards both agents $+20$ upon successfully delivering a dish, which is the return reported in the experimental results. To improve sample efficiency, all algorithms are trained using a shaped reward function that provides each agent an additional reward of 0.1 for picking up an onion, 0.5 for placing an onion in the pot, 0.1 for picking up a plate, and 1.0 for picking up a soup from the pot with a plate. An episode terminates after 400 time steps. Player locations are randomized in each episode. In divided layouts such as AA and FC, we ensure that a player is spawned on each half of the layout.



(a) Cramped Room. (b) Coord. Ring. (c) Forced Coord. (d) Asymmetric Advantages. (e) Counter Circuit.

Figure 10: The five classic Overcooked layouts. Each yellow circle is an onion, while white circles are plates. Grid spaces with multiple yellow (resp. white) circles are onion (resp. plate) piles, which agents must visit to pick up an onion (or plate). The green square is the delivery location, where finished dishes must be sent to receive a reward. Black squares denote free space, while adjacent gray spaces are empty counters. A black pot icon indicates pots, while agents are shown as red and blue pointers. The AHT agent is highlighted.

G Implementation Details

As implementations of prior methods use PyTorch, but this project uses Jax, we re-implemented all methods in this paper, using PPO [50] with Generalized Advantage Estimation (GAE) [49] as

a base RL algorithm and Adam [26] as the default optimizer. An anonymized version of the code is released for reproducibility at https://anonymous.4open.science/r/rotate/, and we recommend consulting it for a full understanding of method implementations. Pseudocode for ROTATE is provided in App. B. This section discusses implementation details such as training time choices, agent architectures, and key hyperparameters for ROTATE and all baselines.

G.1 Training Compute

For fair comparison, all open-ended methods (ROTATE and all variations, PAIRED, Minimax Return) were trained for the same number of open-ended learning iterations and a similar number of environment interactions. For two-stage teammate generation approaches (FCP, BRDiv, CoMeDi), the teammate generation stage is run using a similar amount of compute as the original implementations, while the ego agent training stage is run for a sufficiently large number of steps to allow convergence. We describe the amount of compute used for the teammate generation stage of each baseline below.

In particular, the FCP population is generated by training 22-23 seeds of IPPO with 5 checkpoints per seed for a population of approximately 110 agents—similar to Strouse et al. [53], who trained 32 seeds of IPPO with 3 checkpoints per seed for a population size of 96 agents. On the other hand, BRDiv was trained with a population size of 3-4 agents, until we observed that each agent's learning converged. While we attempted training BRDiv with a larger population size, the algorithm was prone to discovering degenerate solutions where only 2-3 agents in the population could discover solutions with high SP returns, and all other agents in the population would have zero returns. Finally, CoMeDi was trained with a population size of 10 agents, until each agent's learning converged. We attempted to train CoMeDi with a larger population size, but due to the algorithm's quadratic complexity in the population size, its runtime surpassed the available time budget. Nevertheless, the population size of 10 forms a reasonable comparison to ROTATE because (1) the original paper used a population size of 8 for all Overcooked tasks, and (2) the configuration of CoMeDi in this paper runs for a similar wall-clock time as ROTATE.

G.2 Agent Architectures

For all methods considered in this paper, agents are implemented using neural networks and an actor-critic architecture, as is standard for PPO-based RL algorithms. All AHT methods implement policies without parameter sharing [12], to enable greater behavioral diversity. Specifics for ego agents, teammates, and best response agents are described below.

As mentioned in the main paper, ego agents are history-conditioned. Thus, ego agents are implemented with the S5 actor-critic architecture, a recently introduced recurrent architecture shown to have stronger long-term memory than prior types of recurrent architectures. Another advantage of the S5 architecture over typical recurrent architectures (e.g., LSTMs) is that it is parallelizable during training, allowing significant speedups in Jax [31].

On the other hand, teammates and best response agents are state-based. Best response agents are implemented with fully connected neural networks. Teammates are also based on fully connected neural networks, but the precise architecture varies based on the algorithm. For methods where the teammate only interacts with itself (FCP) or with the ego agent (Minimax Return), a standard actor-critic architecture is used. However, for open-ended learning methods that optimize regret (ROTATE and PAIRED), or for teammate generation methods that optimize adversarial diversity (ComeDi and BRDiv), teammates must estimate returns when interacting with multiple agents. Thus, for these methods, the teammate architecture includes a critic for each type of interaction.

In particular, for ROTATE and PAIRED, the teammate must estimate returns when interacting with the ego agent and its best response, and so it maintains a critic network for each partner type. For CoMeDi and BRDiv, given a population with n agents, each teammate must estimate the return when interacting with the other $n - 1$ agents in the population. As it would be impractical to maintain

$n-1$ critics for each teammate, the teammate instead uses a critic that conditions on the agent ID of a candidate partner agent—in effect, implementing the $n-1$ critics via *parameter sharing* [12].

Task	LBF	CR	AA	CC	CoR	FC
Timesteps	**3e5**, 1e6	1e6	1e6	1e6, **3e6**	**3e6**	1e6, **3e6**, 1e7
Number envs	**8**, 16	**8**, 16	8	**8**, 16	8	8
Epochs	7, **15**	15	15	**15**, 30	15	15
Minibatches	**4**, 8	4, 8, **16**, 32	16	16	16	16
Clip-Eps	**0.03**, 0.05	0.03, 0.05, 0.10, 0.15, **0.2**, 0.3	0.2, **0.3**	**0.1**, 0.2	**0.1**, 0.2, 0.3	**0.1**, 0.2
Ent-Coef	5e-3, **0.01**, 0.03, 0.05	5e-3, **0.01**, 0.03, 0.05	**0.01**, 0.02	0.01, 0.03, **0.05**	0.001, 0.01, **0.05**	0.01, **0.05**
LR	1e-4	1e-4	**1e-4**, 1e-3	1e-4, **1e-3**	1e-4, 5e-4, **1e-3**	1e-4, 5e-4, **1e-3**
Anneal LR	**true**, false	**true**, false	true	**true**, false	true	true

Table 1: Hyperparameters for IPPO.

	LBF	CR	AA	CC	CoR	FC
Timesteps	4.5e7	4.5e7	4.5e7	9e7	9e7	9e7
XP Coefficient	**0.1**, 0.75, 1, 10	**1**, 10	10	**0.01**, 10	**0.01**, 10	**0.01**, 0.1, 0.5, 1, 10
Population size	**3**, 4, 5, 10	2, 3, **4**, 5	**3**, 4	**3**, 4	**3**, 4	**3**, 4
Num Envs	8, **32**	8, **32**	8, **32**	8, **32**	8, **32**	8, **32**
LR	1e-4, **5e-4**	1e-4	1e-4	1e-3	1e-3, **5e-4**	1e-3, **5e-4**
Ent-Coef	0.01	0.01	0.01	0.05	0.05	0.05
Clip-Eps	0.03, **0.05**	**0.05**, 0.2	**0.3**	**0.01**, 0.1	0.05, **0.1**	**0.05**, 0.1

Table 2: Hyperparameters for the teammate generation stage of BRDiv.

G.3 Hyperparameters

This section presents the hyperparameters for ROTATE (Table 3), baseline methods (Tables 2 and 4 to 8), and training evaluation teammates with IPPO (Table 1). Note that hyperparameters for the two-stage teammate generation methods are presented in separate tables, where those corresponding to the shared ego agent training stage are presented in Table 4. All experiments in the paper were performed with a discount factor of $\gamma = 0.99$ and $\lambda^{\text{GAE}} = 0.95$.

Hyperparameters were searched for IPPO, BRDiv, and ROTATE, in that order, with the search for earlier methods informing initial hyperparameter values for later methods. Based on prior experience with PPO, we primarily searched the number of environments, epochs, minibatches, learning rate, entropy coefficient, the epsilon used for clipping the PPO objective, and whether to anneal the learning rate. For each hyperparameter, the searched values are listed in the tables, and selected values are bolded. We performed the search manually, typically varying one parameter over the

	LBF	CR	AA	CC	CoR	FC
OEL Iterations	30	30	30	20	20	20
Num Envs	16	16	16	16	16	16
Regret-SP Weight	1, **2**	1, **3**	1, **2**	1, **2**	1, **2**	1, **2**
Minibatches	**4**, 8	8	8	8	8	8
Timesteps per Iter (Ego)	2e6	2e6	2e6	6e6	6e6	6e6
Epochs (Ego)	5, **10**, 20	**10**, 15	10	10	10	**5**, 10
Ent-Coef (Ego)	**1e-4**, 1e-3, 0.01, 0.05	1e-4, **1e-3**, 1e-2	**1e-3**, 0.01	**1e-3**, 0.05	**1e-3**, 0.05	**1e-4**, 1e-3, 1e-2
LR (Ego)	**5e-5**, 1e-4, 1e-3	1e-5, 3e-5, **5e-5**, 1e-4	1e-5, 3e-5, **5e-5,** 1e-4	3e-5, **5e-5,** 1e-3	1e-5, **3e-5**, 5e-5, 1e-3	8e-6, **1e-5**, 3e-5, 5e-5, 1e-4
Eps-Clip (Ego)	0.05, **0.1**	**0.1**, 0.2	**0.1**, 0.3	0.1	0.1	0.1
Anneal LR (Ego)	true, **false**	true, **false**	true, **false**	true, **false**	true, **false**	true, **false**
Timesteps per Iter (T)	1e7	6e6	6e6	1.6e7	1.6e7	1.6e7
Epochs (T)	20	20	20	20	20	20
Ent-Coef (T)	0.05, **0.01**	0.01	0.01	0.05	0.05	0.01, **0.05**
LR (T)	**1e-4**, 1e-3	1e-4	1e-4	1e-3	1e-3	**1e-3**, 1e-4
Clip-Eps (T)	0.1	0.1, **0.2**	0.3	0.1	0.1	**0.1**, 0.2
Anneal LR (T)	**true**, false	**true**, false	false	false	false	true, **false**

Table 3: Hyperparameters for ROTATE. Hyperparameters specific to the teammate training process are marked by "(T)".

	LBF	CR	AA	CC	CoR	FC
Total Timesteps	3e7	3e7	3e7	6e7	6e7	6e7
Num Envs	8	8	8	8	8	8
LR	5e-5	5e-5	5e-5	5e-5	3e-5	1e-5
Epochs	10	10	10	10	10	5
Minibatches	4	4	4	4	4	4
Ent-Coef	1e-4	1e-3	1e-3	1e-3	1e-3	1e-4
Clip-Eps	0.1	0.1	0.1	0.1	0.1	0.1
Anneal LR	false	false	true	true	true	true

Table 4: Hyperparameters for PPO ego agent for all teammate generation methods.

	LBF	CR	AA	CC	CoR	FC
Timesteps Per Agent	2e6	2e6	2e6	4e6	4e6	4e6
Num Seeds	23	23	23	22	22	22
Num Checkpoints	5	5	5	5	5	5
Num Envs	8	8	8	8	8	8
LR	1e-4	1e-4	1e-4	1e-3	1e-3	1e-3
Epochs	15	15	15	15	15	15
Minibatches	4	16	16	16	16	16
Ent-Coef	0.01	0.01	0.01	0.05	0.05	0.05
Eps-Clip	0.03	0.2	0.3	0.1	0.1	0.1
Anneal LR	true	true	true	true	true	true

Table 5: Hyperparameters for teammate generation stage of FCP.

	LBF	CR	AA	CC	CoR	FC
Timesteps Per Iteration	6e6	6e6	6e6	1e7	1e7	1e7
Population Size	10	10	10	10	10	10
Num Envs	16	16	16	16	16	16
LR	5e-4	1e-4	1e-4	1e-3	5e-4	5e-4
Epochs	15	15	15	15	15	15
Minibatches	8	8	8	8	8	8
Ent-Coef	1e-3	0.01	0.01	0.05	0.1	0.01
Eps-Clip	0.05	0.05	0.3	0.01	0.05	0.05
Anneal LR	false	false	false	false	false	false
α	0.2	1.0	1.0	1.0	1.0	1.0
β	0.4	0.5	0.5	0.5	0.5	0.5

Table 6: Hyperparameters for the teammate generation stage of CoMeDi.

	LBF	CR	AA	CC	CoR	FC
Timesteps	7.5e7	7.5e7	7.5e7	1.5e8	1.5e8	1.5e8
Num Seeds	5	5	5	5	5	5
Num Checkpoints	10	10	10	10	10	10
Num Envs	16	16	16	16	16	16
LR	1e-3	1e-4	1e-4	1e-3	1e-3	1e-3
Epochs	15	15	15	15	15	15
Minibatches	4	8	8	8	8	8
Ent-Coef	0.05	0.01	0.01	0.05	0.05	0.05
Eps-Clip	0.1	0.2	0.3	0.1	0.1	0.1
Anneal LR	false	false	false	false	false	false

Table 7: Hyperparameters for PAIRED.

	LBF	CR	AA	CC	CoR	FC
OEL Iterations	30	30	30	20	20	20
Num Envs	16	16	16	16	16	16
Timesteps Per Iter (Ego)	1e6	1e6	1e6	3e6	3e6	3e6
Timesteps Per Iter (T)	1e6	1e6	1e6	3e6	3e6	3e6
LR	1e-4	1e-4	1e-4	1e-3	1e-3	1e-3
Epochs	15	15	15	15	15	15
Minibatches	4	8	8	8	8	8
Ent-Coef	0.01	0.01	0.01	0.05	0.05	0.05
Eps-Clip	0.03	0.2	0.3	0.1	0.1	0.1
Anneal LR	false	false	false	false	false	false

Table 8: Hyperparameters for Minimax Return. Hyperparameters specific to the teammate training process are marked by "(T)".

listed range while holding others fixed, and varying parameters jointly only when varying one at a time did not yield desired results.

Due to compute constraints, hyperparameters for FCP, CoMeDi, PAIRED, and Minimax Return were set based on knowledge of appropriate ranges gained from doing the hyperparameter searches over IPPO, BRDiv, and ROTATE.

H Evaluation Teammate Details

As described in Section 7 of the main paper, evaluation teammates were constructed using three strategies: training IPPO teammates in self-play using varied seeds and reward shaping, training teammates with BRDiv, and manually programming heuristic agents. Note that the evaluation teammates trained using IPPO and BRDiv were trained using different seeds than those used for training ROTATE and baseline methods.

The teammate construction procedure results in distinct teammate archetypes. Generally, IPPO agents execute straightforward, return-maximizing strategies. On the other hand, since BRDiv agents are trained to maximize self-play returns with their best response partner and to minimize cross-play returns with all other best response policies in the population, the generated teammates display more adversarial behavior compared to IPPO and heuristics. Coefficients on the SP and XP returns were carefully tuned to ensure that the behavior was not too adversarial, which we operationalized as teammates where the SP returns were high, but the XP returns were near zero.

Finally, the manually programmed heuristic agents have a large range of skills and levels of determinism. The LBF heuristics are planning-based agents that deterministically attempt to collect the apples in a specific order. Given a best response partner, the LBF heuristics can achieve the optimal task return in LBF. The Overcooked heuristics execute pre-programmed roles that are agnostic to the layout and some basic collision-avoidance logic. The "onion" heuristic collects onions and places them in non-full pots. The "plate" heuristic plates soups that are ready, and delivers them. The "independent" heuristic attempts to fulfill both roles by itself. All three heuristic types have a user-specified parameter that defines the probability that the agent places whatever it is holding on a nearby counter. The feature serves two purposes: first, it creates a larger space of behaviors, and second, it allows the heuristics to work for the FC task, where the agent in the left half of the kitchen must pass onions and plates to the right, while the agent in the right half must pick up resources from the dividing counter, cook soup, and deliver.

Name	Description	Est. BR Return
brdiv_conf1(0)	Teammate trained by BRDiv.	97.396
brdiv_conf1(1)	-	100.0
brdiv_conf1(2)	-	89.583
brdiv_conf2(0)	-	100.0
brdiv_conf2(1)	-	62.5
ippo_mlp(0)	Teammate trained by IPPO to maximize return.	100.0
ippo_mlp_s2c0(2,0)	An intermediate checkpoint of a teammate trained by IPPO to maximize return.	96.354
seq_agent_col	Planning agent that collects food in column-major order (left to right, top to bottom).	100.0
seq_agent_rcol	Planning agent that collects food in reverse column-major order (right to left, bottom to top).	100.0
seq_agent_lexi	Planning agent that collects food in lexicographic order (top to bottom, left to right).	100.0
seq_agent_rlexi	Planning agent that collects food in reverse lexicographic order (bottom to top, right to left).	100.0
seq_agent_nearest	Planning agent that collects food in nearest to farthest order, based on the Manhattan distance from the agent's initial position.	100.0
seq_agent_farthest	Planning agent that collects food in farthest to nearest order, based on the Manhattan distance from the agent's initial position.	100.0

Table 9: Evaluation teammates for LBF and estimated best response returns (percent eaten). Hyphens indicate that the agent description is the same as the previous description.

Name	Description	Est. BR Return
brdiv_conf(0)	Teammate trained by BRDiv.	214.063
brdiv_conf(1)	-	240.940
ippo_mlp(0)	Teammate trained by IPPO to maximize return.	256.875
ippo_mlp(1)	-	253.750
ippo_mlp(2)	-	249.686
independent_agent_0.4	Agent programmed to cook and deliver soups. If holding item, 40% chance of placing item on the counter.	197.188
independent_agent_0	Agent programmed to cook and deliver soups.	132.50
onion_agent_0.1	Agent programmed to place onions in non-full pots. If holding item, 10% chance of placing item on counter.	146.875
plate_agent_0.1	Agent programmed to plate finished soups and deliver. If holding item, 10% chance of placing item on counter.	191.250

Table 10: Evaluation teammates for Cramped Room and estimated best response returns. Hyphens indicate that the agent description is the same as the previous description.

Name	Description	Est. BR Return
brdiv_conf(0)	Teammate trained by BRDiv.	286.875
brdiv_conf(1)	-	335.625
brdiv_conf(2)	-	333.750
ippo_mlp(0)	Teammate trained by IPPO to maximize return.	382.50
ippo_mlp(1)	-	369.375
ippo_mlp(2)	-	312.50
independent_agent_0	Agent programmed to cook and deliver soups.	308.125
onion_agent_0	Agent programmed to place onions in non-full pots.	301.250
plate_agent_0	Agent programmed to place onions in non-full pots.	285.0

Table 11: Evaluation teammates for Asymmetric Advantages and estimated best response returns. Hyphens indicate that the agent description is the same as the previous description.

Name	Description	Est. BR Return
ippo_mlp_cc(0)	Teammate trained by IPPO to maximize return. Navigates counterclockwise around counter.	200.625
ippo_mlp_cc(1)	-	198.120
ippo_mlp_cc(2)	-	194.375
ippo_mlp_pass(0)	Teammate trained by IPPO+reward shaping to pass onions across the counter.	137.813
ippo_mlp_pass(1)	-	103.125
ippo_mlp_pass(2)	-	170.0
independent_agent_0	Agent programmed to cook and deliver soups.	77.189
onion_agent_0.9	Agent programmed to place onions in non-full pots. If holding item, 90% chance of placing item on counter.	80.0
onion_agent_0	Agent programmed to place onions in non-full pots.	81.563
plate_agent_0.9	Agent programmed to plate finished soups and deliver. If holding item, 90% chance of placing item on counter.	97.189
plate_agent_0	Agent programmed to place onions in non-full pots.	76.875

Table 12: Evaluation teammates for Counter Circuit and estimated best response returns. Hyphens indicate that the agent description is the same as the previous description.

Name	Description	Est. BR Return
brdiv_conf1(1)	Teammate trained by BRDiv.	161.250
brdiv_conf1(2)	-	183.440
brdiv_conf2(0)	-	142.810
ippo_mlp(1)	Teammate trained by IPPO to maximize return.	249.688
ippo_mlp(2)	-	246.560
ippo_mlp(3)	-	246.560
independent_agent_0	Agent programmed to cook and deliver soups.	136.250
onion_agent_0	Agent programmed to place onions in non-full pots.	72.50
plate_agent_0	Agent programmed to place onions in non-full pots.	110.938

Table 13: Evaluation teammates for Coordination Ring and estimated best response returns. Hyphens indicate that the agent description is the same as the previous description.

Name	Description	Est. BR Return
brdiv_conf1(0)	Teammate trained by BRDiv.	131.560
brdiv_conf1(2)	-	184.690
brdiv_conf2(1)	-	143.750
brdiv_conf3(0)	-	71.250
brdiv_conf3(2)	-	174.690
ippo_mlp(0)	Teammate trained by IPPO to maximize return.	220.0
ippo_mlp(1)	-	214.380
ippo_mlp(2)	-	225.620
independent_agent_0.6	Agent programmed to cook and deliver soups. If holding item, 60% chance of placing item on the counter.	81.250

Table 14: Evaluation teammates for Forced Coordination and estimated best response returns. Hyphens indicate that the agent description is the same as the previous description.

Descriptions of the evaluation teammates for each task and estimated best response returns are provided in Tables 9 to 14.

Evaluation Return Normalization Details. The lower return bound is set to zero since a poor teammate could always cause a zero return in all tasks considered. Ideally, the upper return bounds would be the returns achieved with the theoretically optimal best response teammate for each evaluation teammate. To approximate this, we instead set the upper bound equal to the maximum average return achieved by any method, for each evaluation teammate.

As described in Section 7, our normalized return metric is similar to the BRProx metric recommended by Wang et al. [59]. The main difference is that we aggregate results using the mean rather than the interquartile mean (IQM), due to challenges around determining appropriate upper bounds for return normalization. In particular, during method development, we used looser BR return estimates to perform return normalization, leading to normalized returns often surpassing 1.0 for certain teammates. Under such conditions, aggregating results using the IQM led to entirely dropping results corresponding to particular teammates.

I Compute infrastructure

Experiments were performed on two servers, each with the following specifications:

- **CPUs**: two Intel(R) Xeon(R) Gold 6342 CPUs, each with 24 cores and two threads per core.
- **GPUs**: four NVIDIA A100 GPUs, each with 81920 MiB VRAM.

The experiments in this paper were implemented in Jax and parallelized across seeds. On the servers above, each method took approximately 4-6 hours of wall-clock time to run.