APPENDIX

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

A 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}

A.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.

A.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, XSP, and SXP interaction data gathered in Lines 5 to 8, 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, and an XSP interaction analogously requires resetting to an SP 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 (and similarly for XSP). Experiences from SP, XP, SXP, and XSP interaction are stored in buffers $D_{\text{SP}}, D_{\text{XP}}, D_{\text{SXP}}, D_{\text{XSP}}$ in the form of a collection of tuples, $D = \langle (s_k, a_k, r_k, s'_k) \rangle_{k=1}^{|D|}$. Lines 12 to 22 of Algorithm 2 then highlight how we use the stored experiences to compute loss functions that the trained models optimize.

Lines 12 and 13 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 explo-

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**$(\pi_{\theta^{\text{ego}}}, \pi_{\theta^{-i}}, p_0^{\text{Env}})$
  $s_{\text{XP}}, s_{\text{SP}} \leftarrow$ **SampleStates**(D_{XP}), **SampleStates**(D_{SP}) ▷ Sample XP states
  $D_{\text{SXP}} \leftarrow$ **Interact**$(\pi_{\theta^{\text{BR}}}, \pi_{\theta^{-i}}, \mathcal{U}(s_{\text{XP}}))$
  $D_{\text{XSP}} \leftarrow$ **Interact**$(\pi_{\theta^{\text{ego}}}, \pi_{\theta^{-i}}, \mathcal{U}(s_{\text{SP}}))$ ▷ Gather SP, XP, SXP, and XSP 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** $\left(\theta^{-i}, \theta^{-i}_{\text{old}}, \sigma^{-i,\text{BR}}_{\text{old}}, D_{\text{SXP}}, \epsilon\right)$
    $L_{\text{reg}}(\theta^{-i}) \leftarrow$ **POL_LOSS_REG_TARG** $\left(\theta^{-i}, \theta^{-i}_{\text{old}}, \sigma^{-i,\text{BR}}_{\text{old}}, \sigma^{-i,\text{ego}}_{\text{old}}, D_{\text{SP}} \cup D_{\text{XP}}, \epsilon\right)$
    $L_V(\sigma^{\text{BR}}) \leftarrow$ **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** $\left(\sigma^{-i,\text{ego}}, \sigma^{-i,\text{ego}}_{\text{old}}, D_{\text{XP}} \cup D_{\text{XSP}}\right)$
    $\theta^{\text{BR}} \leftarrow$ **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
$\text{B}_\pi \leftarrow \text{B}_\pi \cup \langle \theta^{-i} \rangle$ ▷ Add generated teammate policy parameter
Return B_π

ration:

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

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 (Schulman et al., 2015), $A^{\text{GAE}}_{\sigma_{\text{old}}}$. Meanwhile, Line 14 shows how the teammate policy is trained to maximize the ego agent's *regret* based on experiences from XP and SP interactions. 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. The regret-based target function is defined differently but symmetrically for SP and XP states. We describe the target function for regret from **XP** states below, and refer the reader to the code for the target function for regret from SP states.

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

Rather than optimizing a regret function that requires explicitly computing the return-to-go, **POL_LOSS_REG_TARG** *estimates* the XP return via a 1-step bootstrapped return using the teammate critic parameterized by $\sigma^{-i,\text{ego}}$. Similarly, the SP return is estimated using the teammate critic network parameterized by $\sigma^{-i,\text{BR}}$. This results in a regret optimization method that uses the log-derivative trick to optimize objective functions (Williams, 1992; Glynn, 1990). The ROTATE regret estimation method and alternative approaches to maximize regret are further discussed in App. C.2.

Lines 15 to 17 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 18 to 22. In practice, our implementation uses the ADAM optimization technique (Kingma & Ba, 2015). At the end of this teammate generation process, Lines 25 and 26 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 (Zintgraf et al., 2021; Rahman et al., 2021; Papoudakis et al., 2021), π^{ego} and V^{ego} are conditioned on the ego agent's observation-action history to facilitate an adaptive π^{ego} through an improved characterization of teammates' policies. The history-conditioned ego architecture and other practical implementation details are described in App. E. 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.

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}

B 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. E.

PAIRED (Dennis et al., 2020): 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 (Morimoto & Doya, 2005; Villin et al., 2025): 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 (Strouse et al., 2021): A two-stage AHT algorithm where a pool of teammates is generated by running IPPO (Yu et al., 2022) with varying seeds, and saving multiple checkpoints to the pool. The ego agent is an IPPO agent that is trained against the pool.

BRDiv (Rahman et al., 2023): 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 (Charakorn et al., 2023) 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 $1 + 2 * \texttt{xp\_weight}$, 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 (Sarkar et al., 2023): 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. C.1.

B.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 (Rahman et al., 2023) and LIPO (Charakorn et al., 2023) 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.

C 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. Fourth, we examine whether the population generated by ROTATE is useful for training an independent ego agent. Fifth, we present and describe radar charts breaking down the performance of ROTATE on all six benchmark tasks presented in the main paper, and the learning curves for all variants of ROTATE that are tested in this paper. Finally, we present a human proxy evaluation of ROTATE and CoMeDi on the Overcooked tasks.

C.1 Discussion of CoMeDi and Mixed Play

As previously described in App. B, CoMeDi (Sarkar et al., 2023) 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},

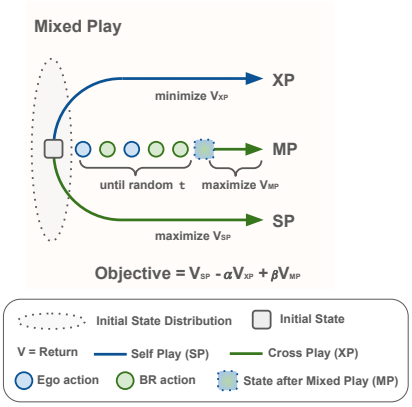


Figure 5: CoMeDi-style mixed-play objective for teammate generation, in the context of open-ended AHT.

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. 8) 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. (2023) 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 (Sarkar et al., 2023) 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} (i.e., SXP states) 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 6a 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 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.

C.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 14 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 regret target function to train the regret-maximizing teammate policy on XP states, with an analogously defined target function for SP states:

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

We maximize regret in states sampled from p_{XP} and p_{SP} 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. Our discussion here focuses on computing Eq. 19 for brevity. However, a similar approach can be used to train a critic network to estimate regret in SP states accurately. The only difference lies in the use of states sampled from p_{SP} and p_{XSP} for training the critic network.

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 XP, as shown in Line 16 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 17 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:

$$\mathbb{E}_{s_t \sim 0.5p_{\text{XP}}+0.5p_{\text{SP}}}\left[\underbrace{\mathbb{E}_{a_{t'}\sim[\text{BR}(\pi^{-i}),\pi^{-i}],P}\left[\sum_{t'=t}^{\infty}\gamma^{t'}r_{t'}\middle|s_t\right]}_{\text{SP return estimate}} - \underbrace{\mathbb{E}_{a_{t'}\sim[\pi^{\text{ego}},\pi^{-i}],P}\left[\sum_{l=0}^{\infty}\gamma^{t'}r_{t'}\middle|s_t\right]}_{\text{XP return estimate}}\right]. \tag{20}$$

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 (Schulman et al., 2015) based on SP and XP interactions. This results in the maximization of the following target function during the teammate policy updates:

$$\mathbb{E}_{s_t \sim 0.5p_{\text{XP}}+0.5p_{\text{SP}}}\left[\underbrace{\mathbb{E}_{a_{t'}\sim[\text{BR}(\pi^{-i}),\pi^{-i}],P}\left[\underbrace{\sum_{t'=t}^{\infty}(\gamma\lambda)^{t'}\,\delta_{t'}^{-i,\text{BR}}}_{\text{GAE}}\middle|s_0\right]}_{\text{SP return estimate}} - \underbrace{\mathbb{E}_{a_{t'}\sim[\pi^{\text{ego}},\pi^{-i}],P}\left[\underbrace{\sum_{t'=t}^{\infty}(\gamma\lambda)^{t'}\,\delta_{t'}^{-i,\text{ego}}}_{\text{GAE}}\middle|s_0\right]}_{\text{XP return estimate}}\right], \tag{21}$$

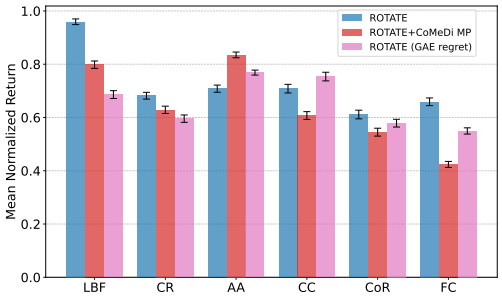
where we define $\delta_t^{-i,\text{BR}}$ and $\delta_t^{-i,\text{ego}}$ as:

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

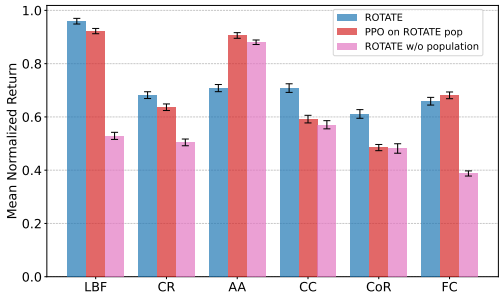
We refer to an instance of the ROTATE algorithm that maximizes regret using this target function as ROTATE with *GAE per-state regret*. In practice, we collect data for SP GAE maximization and XP GAE minimization by first independently sampling two collections of states from D_{SXP} and D_{XP} respectively. 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 (Schulman et al., 2015), effectively providing a different bias-variance tradeoff compared to the previously mentioned methods.

C.3 Experimental Comparisons of ROTATE Teammate Generation Objectives

Figure 6a 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 CoMeDi (Sarkar et al., 2023). 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}.



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



(b) ROTATE compared to an independently trained ego agent on ROTATE's population and an ablation of ROTATE without the population.

Figure 6: The mean normalized returns of ROTATE and various ablations designed to evaluate the effectiveness of ROTATE's regret-based teammate generation objective and population-based ego agent training procedure.

C.4 Training an Independent Ego Agent on the ROTATE Population

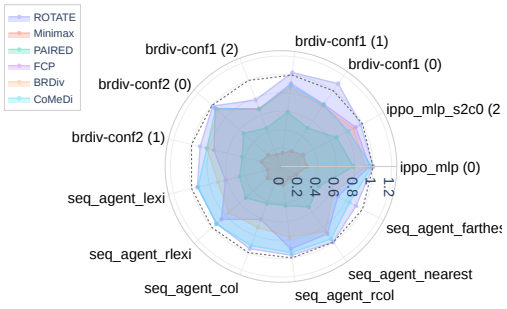
Two-stage AHT algorithms first generate a population of teammates, and next train an ego agent against the population. Although ROTATE's teammate generation mechanism relies on the learning process of a particular ego agent, here, we investigate whether the population generated by ROTATE is useful for training independently generated ego agents. Fig. 6b 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 trained ego agent, while in two tasks, the two ego agents perform similarly (LBF and FC). This result 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.

C.5 ROTATE vs Baselines—Radar Charts

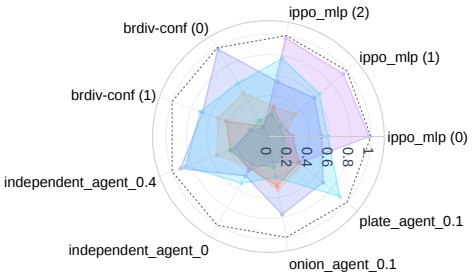
We break down the performance of ROTATE and all baseline methods by individual evaluation teammate policies as radar charts in Fig. 7. The radar charts show that ROTATE achieves higher performance across a larger number and variety of evaluation teammates than baselines. The best baseline, CoMeDi, achieves 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).

C.6 Learning Curves

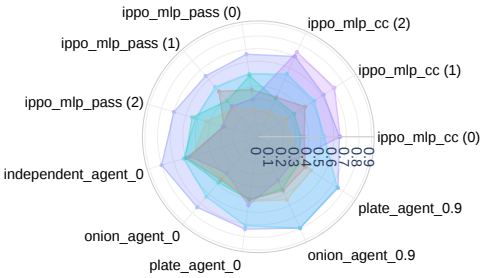
Figure 8 shows learning curves for ROTATE and all ROTATE variations tested in this paper, where the x-axis is the open-ended learning iteration, and the y-axis corresponds to the mean evaluation



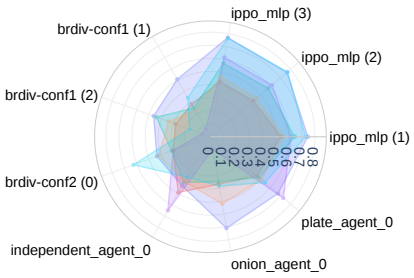
(a) LBF.



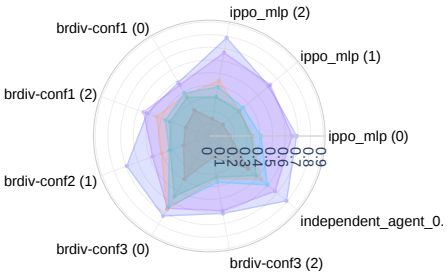
(b) Cramped Room.



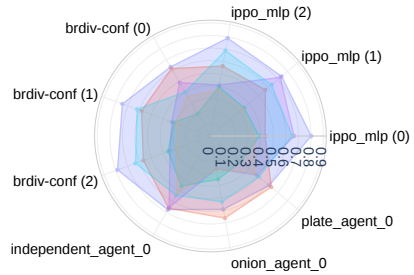
(c) Counter Circuit.



(d) Coordination Ring.



(e) Forced Coordination.



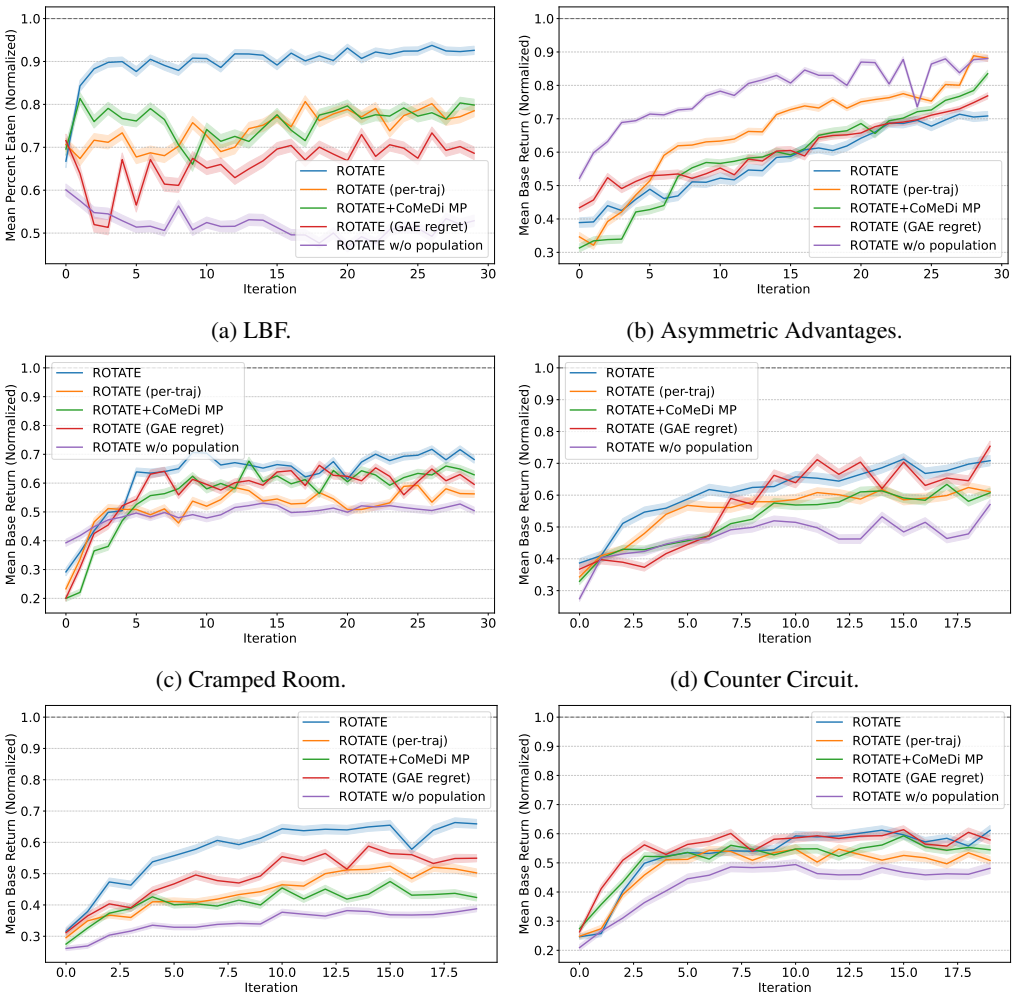
(f) Asymmetric Advantages.

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.

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.

C.7 Human Proxy Evaluations

The results in the main paper show that ROTATE is better able to generalize to unseen partners compared to baseline methods. Here, we evaluate the ability of ROTATE to generalize to *human* partners, compared to the best baseline, CoMeDi. The evaluation is conducted with human proxy teammates generated by behavior cloning on the human gameplay dataset published by (Carroll et al., 2019). Table 2 shows that ROTATE achieves higher returns in coordination with the human proxy compared to CoMeDi, across all five Overcooked layouts.



(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.

	CR	AA	CC	CoR	FC
ROTATE	**0.81** (0.78, 0.84)	**0.66** (0.63, 0.69)	**1.09** (1.05, 1.14)	**0.73** (0.71, 0.75)	**0.62** (0.57, 0.66)
CoMeDi	0.75 (0.71, 0.78)	0.57 (0.54, 0.59)	0.89 (0.83, 0.95)	0.56 (0.54, 0.59)	0.40 (0.38, 0.43)

Table 2: ROTATE outperforms CoMeDi with the human proxy teammate on all Overcooked layouts. Normalized returns and bootstrapped 95% CI's are shown, where the normalization is performed using the human proxy agent's self-play returns.

D Experimental Tasks

Experiments in the main paper are conducted on Jax re-implementations of Level-Based Foraging (LBF) (Albrecht & Ramamoorthy, 2013; Bonnet et al., 2023), and five tasks from the Overcooked suite—Cramped Room (CR), Asymmetric Advantages (AA), Counter Circuit (CC), Coordination Ring (CoR), and Forced Coordination (FC) (Carroll et al., 2019; Rutherford et al., 2024b). Each task is described below.

Level-Based Foraging (LBF) Originally introduced by Albrecht & Ramamoorthy (2013), 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 *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. (2023), which was based on the implementation by Christianos et al. (2020). 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. (2019), 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. (2019) introduced Overcooked to study human-AI coordination, Overcooked has become popularized for AHT research as well (Charakorn et al., 2023; Sarkar et al., 2023; Erlebach & Cook, 2024).

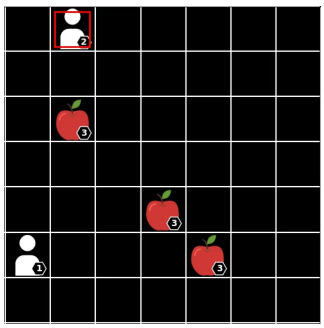
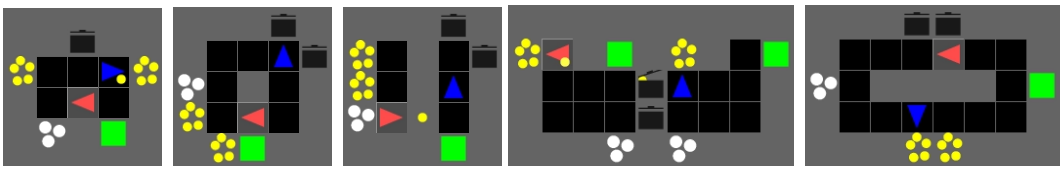


Figure 9: Level-based foraging environment. The apple icons denote food. The number on the icon indicates each player's and food's level. The AHT player is indicated by the red box.

We use the Jax re-implementation of the Overcooked suite by Rutherford et al. (2024b), which is based on the original implementation by Carroll et al. (2019). 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.

E Implementation Details

As implementations of prior methods use PyTorch, but this project uses Jax, we re-implemented all methods in this paper, using PPO (Schulman et al., 2017) with Generalized Advantage Estimation (GAE) (Schulman et al., 2015) as a base RL algorithm and Adam (Kingma & Ba, 2015) 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. A. This section discusses implementation details such as training time choices, agent architectures, and key hyperparameters for ROTATE and all baselines.

E.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. (2021), 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.

E.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 (Christianos et al., 2020), 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 (Lu et al., 2023).

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* (Christianos et al., 2020).

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 3: 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 4: Hyperparameters for the teammate generation stage of BRDiv.

	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 5: 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 6: 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 7: 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 8: 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 9: 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 10: Hyperparameters for Minimax Return. Hyperparameters specific to the teammate training process are marked by "(T)".

E.3 Hyperparameters

This section presents the hyperparameters for ROTATE (Table 5), baseline methods (Tables 4 and 6 to 10), and training evaluation teammates with IPPO (Table 3). 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 6. 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 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.

F 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

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 11: 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 12: 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 13: 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 14: 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 15: 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 16: Evaluation teammates for Forced Coordination and estimated best response returns. Hyphens indicate that the agent description is the same as the previous description.

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.

Descriptions of the evaluation teammates for each task and estimated best response returns are provided in Tables 11 to 16.

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. (2024b). 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.

G 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.