# BEYOND FIXED COORDINATION GRAPHS: LEARNING INTERACTION TOPOLOGIES IN OPEN AD HOC TEAMWORK

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

In ad hoc teamwork, the goal is to enable an autonomous agent to collaborate with other unknown agents without prior coordination or joint training. We study an open-environment variant of ad hoc teamwork in which agents with fixed policies can enter and exit the system at any time. Modeling multi-agent interaction is central to this problem, and prior coordination-graph methods rely on fixed topologies — dense complete graphs or sparse structures such as star graphs — often paired with cooperative-specific regularizers. These structural priors are efficient within the setting they are designed for, but degrade sharply outside it: dense fixed topologies accumulate noise as team size grows, while cooperative regularizers break down in the presence of adversarial agents. We introduce a dynamic, attention-based approach to model multi-agent interaction that requires no structural inductive bias or domain-specific regularizer, and whose learned interaction weights yield an interpretable coordination graph. We evaluate our method on both purely cooperative and mixed cooperative-competitive environments: it matches or exceeds the baselines on different benchmark problems, remains the only method that is competitive across both and scales substantially better as team size grows in the presence of adversarial agents. Our method is also highly robust to behavioral uncertainty, achieving substantial improvements in zero-shot generalization to unseen teammate policy distributions and team sizes.

## 1 INTRODUCTION

Recently, there have been many exciting advances in the field of multi-agent RL (MARL) Foerster et al. (2018); Rashid et al. (2020); Lowe et al. (2017); Sunehag et al. (2017). Most methods in MARL follow a common learning paradigm of centralized training, where all agents' policies are trained together. This allows the agents to learn from each other while improving their performance. However, in many real-world problems, agents may have to interact with other unfamiliar agents that were not trained together. For example, in a multi-agent self-driving cars domain, multiple agents are trained jointly in a traffic simulation. When such an agent's policy is deployed in a real-world traffic environment, it has to interact with other unfamiliar autonomous car agents (or even human agents). This is a much more challenging problem than the standard MARL setting. In this scenario, the agent must quickly adapt and collaborate with unfamiliar agents to achieve a common goal. Stone et al. (2010) introduced the concept of ad hoc teamwork, in which the goal is to develop methodologies for a *learner-agent* that can adapt and collaborate effectively with other unknown agents without any prior coordination or joint training. The *open-environment* setting called open ad hoc teamwork (OAHT) Rahman et al. (2021); Mirsky et al. (2022) is a variant of ad hoc teamwork in which agents can enter and exit the environment at any time step. This makes the environment highly dynamic and uncertain, hindering the *learner-agent* from cooperating effectively with other teammates.

Since OAHT is a cooperative environment setup, one of the key challenges is the multi-agent credit assignment problem, which involves determining how the *learner-agent*'s action affects the overall team reward. Additionally, because the *learner-agent* is unfamiliar with other agents, it is crucial to identify which agents to coordinate with at any time step. This can be achieved by learning an interaction graph in which nodes represent agents and edges represent interactions.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

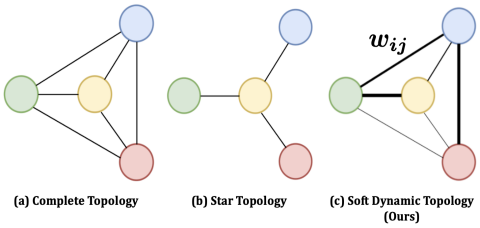



**Figure 1:** Different interaction topologies.

The coordination graph (CG) Guestrin et al. (2002); Boehmer et al. (2020) is a common technique in the cooperative MARL literature for modeling interactions among agents. The topology of CG is then used to decompose the joint-action value function, thereby addressing the multi-agent credit assignment problem in a cooperative environment. Previous methods Rahman et al. (2021); Wang et al. (2024); Sun et al. (2024) relied on a fixed coordination graph approach to model interactions among agents, either through dense or strictly constrained inductive bias on the topology. The rigidity of the graph structure limits the scalability and *learner-agent*'s ability to generalize under behavioral uncertainty of other agents, as it cannot dynamically identify the most critical interactions.

In this work, we introduce a dynamic approach to model the multi-agent interaction using an attention mechanism Vaswani et al. (2017). We use a transformer-based encoder to learn contextualized embeddings that capture complex inter-agent relationships in the OAHT environment. The embedding is then used as input for the joint action-value function computation and agent behavior model. As shown in Figure 1, rather than relying on a predefined interaction structure, our method continuously evaluates and adjusts the interaction strengths between the *learner-agent* and all other entities in the coordination graph. This attention-driven approach allows the agent to implicitly learn which teammates or adversaries are most relevant to its current action-value estimation. We summarize our key contributions as follows:

1. We propose a novel soft coordination graph framework for the OAHT problem using an attention mechanism. It enables the *learner-agent* to dynamically evaluate the relevance of interactions with other agents and learn rich representation embeddings for effective team collaboration. Our proposed method is also general for both cooperative and mixed environments; it does not require any environment-specific regularizer.
2. We evaluate our method on standard benchmark problems for both cooperative and mixed environments. Our method is scalable to larger team sizes, as we empirically demonstrate significant improvement over the SOTA baselines.
3. Our method is highly robust to behavioral uncertainty, achieving substantial improvements in zero-shot generalization over unseen teammate policy distributions. Additionally, the interaction weights provide a meaningful interpretation of agent interactions, which is not possible with previous methods.

## 2 Related Work

**Open ad hoc teamwork:** Several existing works on ad hoc teamwork Stone et al. (2010); Agmon & Stone (2012); Ribeiro et al. (2022); Barrett et al. (2017) have mainly focused on the closed-environment setting in which the team size is fixed. Rahman et al. (2021) (GPL) introduced the *open-environment* variant of the problem. They propose a graph-based policy learning, which uses a graph neural network (GNN) with a complete graph topology to model interactions among agents. However, the complete graph structure results in noisier interactions and does not effectively identify the relevant interactions for the *learner-agent*. Similarly, Wang et al. (2024) (CIAO), a purely cooperative approach, introduces cooperative game-theoretic regularizers to enforce stable interactions on two graph topologies — star and complete. They also kept the topology fixed as a strict inductive bias. While the structural inductive biases succeed in purely cooperative settings, they expose a critical vulnerability: their proposed regularizers assume all nodes in the graph seek to maximize social welfare. In mixed environments with adversarial agents, forcing the cooperative regularizers onto them results in suboptimal performance. Sun et al. (2024) (Open-OTAF) learns a latent *teamwork situation* by clustering teammates by behavior and encoding the team as a heterogeneous graph. However, the graph still connects every pair of agents, and it builds on the same fully connected coordination graph as GPL, thereby inheriting its complete-graph limitation.

**Attention Mechanism:** The attention mechanism introduced by Vaswani et al. (2017) has been widely adopted in standard multi-agent RL approaches. Iqbal & Sha (2019) used it to select relevant observation information specific to each agent. Meng et al. (2023) adapted the Decision Transformer

Chen et al. (2021) for multi-agent RL, while Zhao & Xie (2025) addressed the credit assignment problem using the attention-based framework. Meanwhile, Chen et al. (2020) proposed an attention framework focused on ad hoc teamwork scenarios. The attention network is used to learn an embedding of a sequence of states and map it to an agent's actions. Zhang et al. (2025) introduced a goal-based Decision Transformer that learns teammates' rewards-to-go and subgoals, for an offline ad hoc teamwork. In contrast to previous methods, we use attention to learn a rich embedding that effectively captures agent interactions in a highly dynamic *open-environment* variant of the ad hoc teamwork problem. Furthermore, the attention weights are used to compute the interaction strength in the coordination graph.

## 3 BACKGROUND

An open ad hoc teamwork can be formalized as an open stochastic Bayesian game (OSBG) $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, R, P \rangle$ Rahman et al. (2021), where $\mathcal{N}$ is the agent set, and $\mathcal{S}$ and $\mathcal{A}$ are the state and action spaces, respectively. We assume all agents share the same state and action space. We define the joint-action space as $\mathcal{A}_{\mathcal{N}} = \{\boldsymbol{a} | \boldsymbol{a} \in \mathcal{P}(\mathcal{N} \times \mathcal{A}), \forall (i, a^i), (j, a^j) \in \boldsymbol{a} : i = j \implies a^i = a^j\}$, where $\mathcal{P}$ denotes the power set. $r : \mathcal{S} \times \mathcal{A}_{\mathcal{N}} \to \mathbb{R}$ denotes the reward function, and $P : S \times \mathcal{A}_{\mathcal{N}} \to \Delta(S)$ is the transition function which defines the probability of next state given the current state and joint actions and $\Delta(X)$ denotes all probability distributions over $X$. An OSBG game starts with an initial environment state $s_0 \sim p(s_0)$, which also includes an initial set of agents $\mathcal{N}_0$. At each time-step $t$ with $\mathcal{N}_t$ agents, each agent $j \in \mathcal{N}_t$ takes an action $a_t^j$ following a policy $\pi^j : \mathcal{S} \times \mathcal{A} \to [0, 1]$ resulting in joint action $\boldsymbol{a}_t = \{a_t^j | j \in \mathcal{N}_t\}$. The environment then transitions to the next state $s_{t+1}$ (with the next set of agents $\mathcal{N}_{t+1}$) and a reward $r(s_t, \boldsymbol{a}_t)$. Let $i$ denote the *learner-agent*. The main objective in OAHT is to find the policy $\pi^i$ of the *learner-agent* that maximizes:

$$Q_{\pi^i}(s_0, a_0^i) = \mathbb{E}_{s_t, a_t^i \sim \pi^i, \boldsymbol{a}_t^{-i} \sim \boldsymbol{\pi}^{-i}} \Big[ \sum_{t=0}^{\infty} \gamma^t \cdot \; r\big(s_t, \langle a_t^i, \boldsymbol{a}_t^{-i} \rangle\big) \big| s_0, a_0^i, \pi^i \Big] \tag{1}$$

Where $Q_{\pi^i}(s, a^i)$ is local action-value function of *learner-agent*'s policy $\pi^i$. $\boldsymbol{a}_t^{-i} = \boldsymbol{a}_t \setminus a_t^i$, $\boldsymbol{\pi}^{-i}$ denotes the joint policy of agents excluding the *learner-agent* policy. In a typical OAHT setting, other agents can enter and leave the system without any notifications, so the *learner-agent* does not have access to the policies of other agents $\boldsymbol{\pi}^{-i}$. But, it can observe the actions $\boldsymbol{a}^{-i}$ taken by other agents. $r(s_t, \boldsymbol{a}_t)$ is the *learner-agent*'s reward, which also depends on the actions of other agents. The joint action-value function is defined as:

$$Q_{\boldsymbol{\pi}}(s, \boldsymbol{a}) = \mathbb{E}_{s_t, a_t^i \sim \pi^i, \boldsymbol{a}_t^{-i} \sim \boldsymbol{\pi}^{-i}} \Big[ \sum_{t=0}^{\infty} \gamma^t \cdot \; r\big(s_t, \langle a_t^i, \boldsymbol{a}_t^{-i} \rangle\big) \Big| s_0 = s, \boldsymbol{a}_0 = \boldsymbol{a}, \boldsymbol{\pi} \Big] \tag{2}$$

A naïve approach to learn the *learner-agent*'s Q-function $Q_{\pi^i}(s, a^i)$ is that of an independent learning paradigm. For example, in independent Q-learning (IQL) Tan (1993), the agent assumes it is acting alone in the environment by discounting the effects of other agents' actions. However, the policy trained with this action-value function faces non-stationarity and multi-agent credit assignment problems since it is trained with the team's reward signal. The non-stationarity problem is also exacerbated in OAHT due to the *open-environment* setting. A widely used centralized training and decentralized execution (CTDE) approach Oliehoek et al. (2008), in which a global critic is learned, helps alleviate the non-stationarity issue.

## 4 METHOD

In OAHT, since the *learner-agent* does not know the policies of other agents, its action-value function in (1) must account for uncertainties caused by other agents' actions. This is computed by marginalizing over the joint action-value function of the other agents' actions.

$$\bar{Q}_{\pi^i}(s, a^i) = \mathbb{E}_{\boldsymbol{a}_t^{-i} \sim \boldsymbol{\pi}^{-i}} \Big[ \mathbb{E}_{s_t, a_t^i \sim \pi^i} \Big[ \sum_{t=0}^{\infty} \gamma^t \cdot \; r\big(s_t, \langle a_t^i, \boldsymbol{a}_t^{-i} \rangle\big) \big| s_0 = s, a_0^i = a^i, \pi^i \Big] \Big] \tag{3}$$

$$= \mathbb{E}_{\boldsymbol{a}^{-i} \sim \boldsymbol{\pi}^{-i}} \big[ Q_{\pi^i}(s, \langle a^i, \boldsymbol{a}^{-i} \rangle) \big] \tag{4}$$

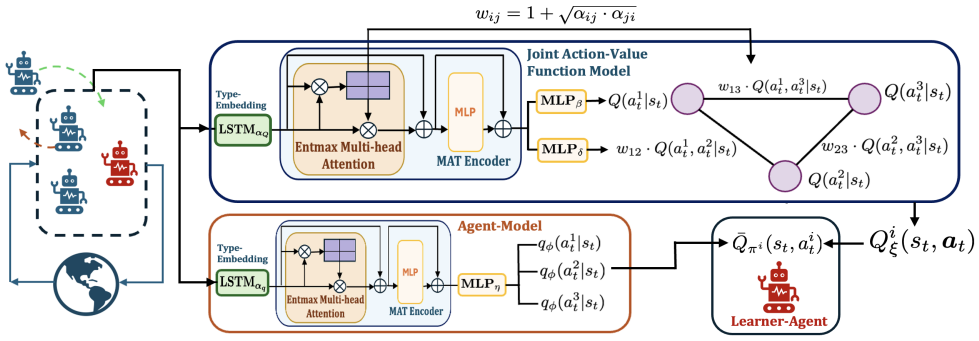



**Figure 2:** The architecture consists of two main components: the joint action-value function model and the agent model. The LSTM processes a sequence of observations to generate type-embeddings for each agent. These embeddings are fed into the MAT encoder, producing contextualized representations of the interaction, which are then used to compute utilities in the coordination graph and the agent behavior in the agent model component. The edge weights $w_{ij}$, along with the utilities, are used in joint action-value function decomposition. The action-value function of the *learner-agent* is computed by marginalizing over the joint action-value function of the other agents' actions using the agent model.

Where $Q_{\pi^i}(s, \langle a^i, \boldsymbol{a}^{-i} \rangle)$ is *learner-agent*'s estimate of the joint action-value function given $\boldsymbol{a}^{-i}$. Although the *learner-agent* doesn't have access to $\boldsymbol{\pi}^{-i}$, it can observe the actions taken by other agents. We learn a common agent behavior model $q_\phi(a_j|s)$ for other agents via maximum-likelihood estimation. We then approximate the above expectation using the learned agent model:

$$\bar{Q}_{\pi^i}(s, a^i) \approx \mathbb{E}_{\boldsymbol{a}^{-i} \sim \boldsymbol{q}_\phi(\cdot|s)} \left[ Q_{\pi^i}(s, \langle a^i, \boldsymbol{a}^{-i} \rangle) \right] \tag{5}$$

Where $\boldsymbol{q}_\phi(\boldsymbol{a}^{-i}|s) = \prod_{j \neq i} q_\phi(a^j|s)$. Another key challenge in OAHT is the credit assignment problem, since the *learner-agent* also has to maximize the team reward. Implicit credit-assignment techniques in Sunehag et al. (2017); Rashid et al. (2020) involve breaking down the joint action value function into its individual local value function components. Each agent's policy is then trained using its corresponding local value function. However, these techniques do not adequately capture the interactions among agents. In contrast, the coordination graph (CG) Guestrin et al. (2002); Boehmer et al. (2020) provides a more robust method for decomposing the joint action-value function by representing it as the sum of both individual and pairwise utilities.

$$Q_i(s_t, \boldsymbol{a}_t) = \sum_{j \in N_t} Q_j(a_t^j|s_t) + \sum_{j,k \in N_t, j \neq k} Q_{j,k}(a_t^j, a_t^k|s_t), \tag{6}$$

Where $Q_j(a_t^j|s_t)$ represents the utility of an individual agent's action $a_t^j$, and $Q_{j,k}(a_t^j, a_t^k|s_t)$ captures pairwise utility between agents $j$ and $k$. GPL and Open-OTAF use a complete graph topology, shown in Figure 1(a), as their coordination graph; CIAO introduces a star graph topology shown in Figure 1(b), in addition to the complete graph. In all previous approaches, the graph structure remains static, meaning the connection strength between agents does not change over the course of their interactions. Due to the highly dynamic nature of the OAHT environment, a significant limitation of a static CG model is that it may lead to inaccurate estimates of the joint action-value function because of its fixed topology. We introduce a dynamic coordination graph that uses an attention mechanism to effectively model the interactions between *learner-agent* and other agents in the environment. As shown in Figure 1(c), the edge weight is computed based on the current environment scenario for each time step. The weights then adjust the utility terms of the value decomposition, creating a soft, interpretable coordination graph that adapts seamlessly to varying team compositions in both fully cooperative and mixed environments. The overall architecture is illustrated in Figure 2.

**Permutation-invariant embeddings:** We implement the attention mechanism using the multi-agent transformer (MAT), as described in Wen et al. (2022). The transformer architecture is also particularly well-suited to the *open-environment* setting because of its ability to handle variable-length inputs. To effectively manage the varying number of agents within the environment, we omit positional encodings. This approach ensures that the MAT encoder operates on the input as a strictly permutation-invariant, unordered set of entities. The encoder takes as input a sequence of agents'

type embeddings $\langle \theta_t^1, \ldots, \theta_t^{N_t} \rangle$ and generates contextualized representations of the interaction.

$$\langle \bar{\theta}_t^1, \ldots, \bar{\theta}_t^{N_t} \rangle = \mathrm{MAT}_{\psi_Q}(\langle \theta_t^1, \ldots, \theta_t^{N_t} \rangle), \quad \theta_t^i = \mathrm{LSTM}_\alpha(\langle o_{t:t-k}^i \rangle) \tag{7}$$

$\theta_t^i$ denotes the type-embedding of each agent. In ad hoc teamwork, since the *learner-agent* is not jointly trained with other agents, it is unfamiliar with their policies. Therefore, learning the agent types is crucial for modeling their behavior and for representing the joint action-value function. We adapt the GPL's type inference model, in which a type-embedding encoder is implemented using LSTM Hochreiter et al. (1997).

**Attention-based soft topology:** Learning the relevance of other agents in the environment is key in the OAHT problem. An agent on the far side of the map may be irrelevant to the *learner-agent* for its immediate action. In this work, we use the self-attention mechanism as a *dynamic relation filter*, allowing the agent to focus on teammates that significantly impact its immediate goals while ignoring irrelevant ones. The attention weights from MAT serve as a relevance score for interactions between the agents, computed as $\alpha_{ij} = \mathrm{entmax}(\langle q_i \cdot k_j \rangle / \sqrt{d})$, where $q_i, k_j$ and $\sqrt{d}$ are the query, key, and scaling factor, respectively. While the standard softmax function produces dense attention distributions that assign small, non-zero weights to every agent, it effectively recreates the noise bottleneck of a complete graph. In contrast, entmax Peters et al. (2019) yields sparse probability distributions. This sparsity serves as an implicit filtering mechanism, allowing the model to isolate relevant interactions.

Intuitively, the attention weight $\alpha_{ij}$ quantifies how much agent $i$ values agent $j$, indicating a one-way relationship. However, the edge weights in an interaction graph are symmetrical. A strong connection between two agents should exist only when both agents actively attend to one another. Therefore, we compute the edge weight between any two agents as a geometric mean of their reciprocal attention scores $w_{ij} = 1 + \sqrt{\alpha_{ij}\alpha_{ji}}$. This ensures that a strong structural edge forms only if both agents actively attend to one another, thereby preventing unstable, asymmetric dependencies. Furthermore, the additive constant of one acts as a structural residual connection. It stabilizes early training by providing a baseline from a complete-graph prior, which the attention mechanism learns to aggressively modulate as it searches for an effective topology for the coordination graph. We then use the edge weights in joint action-value function decomposition.

$$Q_\xi^i(s_t, \boldsymbol{a}_t) = \sum_{j \in N_t} Q_\beta^j(a_t^j | s_t) + \sum_{j,k \in N_t, j \neq k} w_{jk} \cdot Q_\delta^{j,k}(a_t^j, a_t^k | s_t), \tag{8}$$

**Agent behavior model:** Since the *learner-agent* is unfamiliar with the other agents in the environment, it is essential to learn an agent behavioral model. The agent model is mainly used in the approximation of the *learner-agent*'s local Q-function by marginalizing the actions of other agents from the joint action-value function in (5). For modeling agent behavior, previous methods have used a relational forward model Tacchetti et al. (2019), a graph neural network in which the message passing is done over the complete graph. This results in an embedding where each agent has a uniform contribution to every other agent. On the other hand, a transformer-based encoder with entmax activation effectively filters out irrelevant agents, resulting in a less noisy embedding.

$$q_\phi(a_t^j | s_t) = \mathrm{softmax}(\mathrm{MLP}_\eta(\mathrm{MAT}_{\psi_q}(\langle \theta_t^1, \ldots, \theta_t^{N_t} \rangle))) \tag{9}$$

Assuming conditional independence among the teammates' actions given the current state, the joint action probability distribution is obtained by: $q_\phi(a_t^{-i} | s_t) = \prod_{j \neq i} q_\phi(a_t^j | s_t)$. The agent behavior model is trained by minimizing the negative log-likelihood loss:

$$\mathcal{L}_{\mathrm{agent-model}}(\phi) = \mathbb{E}_{(s, \boldsymbol{a}^{-i}) \sim \mathcal{D}} \left[ -\log q_\phi(\boldsymbol{a}^{-i} | s) \right] = \mathbb{E}_{(s, \boldsymbol{a}^{-i}) \sim \mathcal{D}} \Big[ \sum_{j \neq i} -\log q_\phi(a^j | s) \Big] \tag{10}$$

Where $\phi = \{\eta, \psi_q, \alpha_q\}$. $\eta$, $\psi_q$, and $\alpha_q$ are model parameters for the $\mathrm{MLP}_\eta$, $\mathrm{MAT}_{\psi_q}$ encoder and type-embedding ($\mathrm{LSTM}_{\alpha_q}$) for the agent modeling module, respectively.

**Action value function computation:** We adapt GPL's matrix factorization technique to model both individual and pairwise utility functions as below:

$$Q_\beta^j(a_t^j | s_t) = \mathrm{MLP}_\beta(\langle \bar{\theta}_t^j, \bar{\theta}_t^i \rangle)[a_t^j], \quad Q_\delta^{j,k}(a_t^j, a_t^k | s_t) = \mathrm{MLP}_\delta(\langle \bar{\theta}_t^j, \bar{\theta}_t^i \rangle)^\top \mathrm{MLP}_\delta(\langle \bar{\theta}_t^k, \bar{\theta}_t^i \rangle)[a_t^j, a_t^k] \tag{11}$$

However, the key difference from the GPL's utility models is that we use the transformer-based embeddings $\bar{\theta}_t^j$ as input to the MLPs, rather than the type-embeddings $\theta_t^j$. This provides a better representation of the interactions that helps with learning the utility models more effectively. We estimate the *learner-agent*'s individual value function $\bar{Q}_{\pi^i}(s_t, a_t^i)$ in (5) by marginalizing other agents' action in the coordination graph-based joint utility function in (8).

$$\bar{Q}_{\pi^i}(s_t, a_t^i) = Q_\beta^i(a_t^i|s_t) + \sum_{a^j, j\neq i} \Big(Q_\beta^j(a^j|s_t) + w_{ij} \cdot Q_\delta^{i,j}(a_t^i, a^j|s_t)\Big) q_\phi(a^j|s_t) + \sum_{a^j, a^k, j,k\neq i} w_{jk} \cdot Q_\delta^{j,k}(a^j, a^k|s_t) q_\phi(a^j|s_t) q_\phi(a^k|s_t) \tag{12}$$

Learning the joint-action value function in OAHT is challenging because *learner-agent* does not have access to other agents' policies $\boldsymbol{\pi}^{-i}$. However, since the other agents are pretrained, an increase in *learner-agent*'s individual action-value function will improve the overall joint-action value function. Therefore, we use the following TD loss:

$$\mathcal{L}_{\text{TD}}(\xi) = \mathbb{E}_{(s,\boldsymbol{a},r,s')\sim\mathcal{D}}\Big[\Big(\Big(r + \gamma \max_{a^i} \bar{Q}_{\pi^i}(s', a^i)\Big) - Q_\xi^i(s, \boldsymbol{a})\Big)^2\Big] \tag{13}$$

Where $\xi = \{\delta, \beta, \psi_Q, \alpha_Q\}$. $\delta$, $\beta$, $\psi_Q$, and $\alpha_Q$ are parameters for the $\text{MLP}_\delta$, $\text{MLP}_\beta$, $\text{MAT}_{\psi_Q}$ encoder and type-embedding ($\text{LSTM}_{\alpha_Q}$) for the action-value function module, respectively. The *learner-agent* takes action using the $\epsilon$-greedy policy on its individual Q-function $\bar{Q}_{\pi^i}(s, a^i)$. The pseudocode for the algorithm is provided in the supplementary section C.

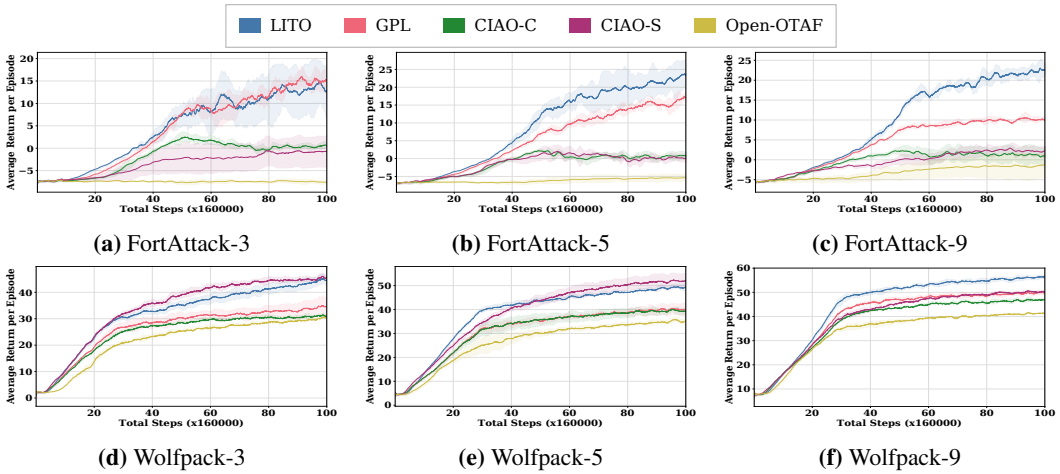
## 5 Experiments

We design our experiments to answer three key questions: (i) Does a dynamically learned topology improve performance and scalability over fixed graph structures? (ii) Generalization: Does it generalize more robustly to the unseen team sizes and teammate behaviors? and (iii) Edge-attention analysis: Does the learned topology exhibit interpretable, semantically meaningful structure? To address these questions, we evaluate on two benchmark OAHT problems: FortAttack and Wolfpack in which agents can enter and exit the system at any time. More details regarding the environments are provided in the supplementary section A. We refer to our method as LITO: Learning Interaction Topologies in Open ad hoc teamwork. We compare against four SOTA baselines: GPL, CIAO-C, CIAO-S, and Open-OTAF.[1] LITO is implemented using the publicly available official CIAO codebase, which also includes the GPL implementation. All experiments are run over 6 seeds, with performance measured as average return per episode over training steps. More details on the experimental setup are provided in the supplementary section D. We first present the performance results on two OAHT benchmark problems.

**FortAttack:** It is a mixed cooperative-competitive setting in which the *learner-agent* is part of a team of pre-trained guard agents protecting a castle against pre-trained adversarial attackers. Figure 3 (a, b, and c) shows the results for maximum team sizes of 3, 5, and 9, respectively. Both variants of CIAO (C and S) struggled to perform across the three settings, as they rely on a rigid cooperative game-theoretic regularizer on top of a fixed topology. In Figure 3(a), GPL and LITO give similar results, but as the maximum team size increases in Figure 3(b and c), LITO perform better than GPL. For a max team size of 9, in Figure 3(c), there are 18 interacting entities (9 guards and 9 attackers), which is a much more challenging setting. LITO scales seamlessly, achieving a significant performance gap over the other baselines.

**Wolfpack:** Next, we evaluate in a purely cooperative setting of the Wolfpack domain. A pack of wolf agents (hunters) in which *learner-agent* is also part of the pack must catch a moving prey; a capture occurs when at least two hunter agents are positioned on adjacent sides of the prey. Figure 3(d, e, and f) shows results for maximum team sizes of 3, 5, and 9, respectively. For a smaller maximum team size of 3 and 5, CIAO-S is superior, with LITO a very close second. In smaller team sizes, the inductive bias of the star graph topology in CIAO-S helps achieve better results. However, for a larger team size of 9, LITO's dynamic topology excels as environmental complexity increases, surpassing all baselines and achieving the highest scores. In a more challenging setting,

[1] Our own implementation as code was not publicly available, and the authors did not respond upon contact.




**(a)** FortAttack-3 **(b)** FortAttack-5 **(c)** FortAttack-9

**(d)** Wolfpack-3 **(e)** Wolfpack-5 **(f)** Wolfpack-9

**Figure 3:** Learning curves for FortAttack and Wolfpack for maximum team sizes of 3, 5, and 9.

**Table 1:** Zero-shot generalization to unseen policy distributions (evaluated at maximum team size 5)

| Env. | GPL | CIAO-C | CIAO-S | Open-OTAF | LITO |
|---|---|---|---|---|---|
| Fort. | $3.86 \pm 1.60$ | $0.35 \pm 0.55$ | $0.43 \pm 0.35$ | $-4.80 \pm 1.14$ | $\mathbf{6.74 \pm 1.96}$ |
| Wolf. | $25.40 \pm 3.66$ | $20.70 \pm 2.96$ | $32.86 \pm 3.82$ | $30.62 \pm 1.37$ | $\mathbf{35.66 \pm 4.75}$ |

LITO beats CIAO, even though CIAO is designed for a purely cooperative setting. These results demonstrate that LITO can dynamically learn relevant interaction weights and a rich contextual embedding of the environment (in both mixed and purely cooperative), thereby improving performance and scaling seamlessly.

### 5.1 Generalization

**Policy distribution generalization:** To test robustness against behavioral uncertainty, all methods are evaluated on other agents' policies drawn from a disjoint policy distribution not seen during training (evaluated at a maximum team size of 5). As shown in Table 1, LITO exhibits superior performance on both domains. Because our attention mechanism dynamically isolates relevant interactions rather than memorizing specific agent trajectories, a behavior explicitly visualized in the attention maps in Figure 4, it excels at generalizing to novel coordination dynamics. In FortAttack, our model achieves a return of 6.74, nearly doubling the performance of GPL (3.86). Most notably, in Wolfpack, our generic model secures the highest average return (35.66) under unseen behaviors, outperforming even the specialized CIAO-S (32.86). This indicates that dynamically learning interaction relevance via attention is superior to static message passing for adapting to novel behaviors.

**Team size generalization:** Table 2 presents the zero-shot generalization performance of models trained on a maximum team size of 3 and evaluated on larger teams of 5 and 9. For each environment and size, we normalize every method's return by the best-performing method in that setting, $s_i = \max(0, x_i / \max_j x_j)$, clipping negative returns to zero to reflect structural failure rather than partial credit. The WCRP column reports each method's minimum normalized score across all four settings. While GPL, CIAO-C, and CIAO-S each achieve the best raw score in at least one setting, all three collapse in at least one other setting (GPL in Wolfpack; both CIAO variants in FortAttack, where returns turn negative). Our method never leads outright, but consistently stays within a bounded margin of the best method in every

**Table 2:** Worst-case relative performance (WCRP) across team-size generalization settings.

| Method | Fort.5 | Fort.9 | Wolf.5 | Wolf.9 | WCRP |
|---|---|---|---|---|---|
| GPL | 1.00 | 1.00 | 0.68 | 0.65 | 0.65 |
| CIAO-C | 0.01 | 0.00 | 0.82 | 0.94 | 0.00 |
| CIAO-S | 0.00 | 0.00 | 1.00 | 1.00 | 0.00 |
| Open-OTAF | 0.00 | 0.00 | 0.71 | 0.71 | 0.00 |
| LITO | 0.87 | 0.84 | 0.87 | 0.84 | **0.84** |

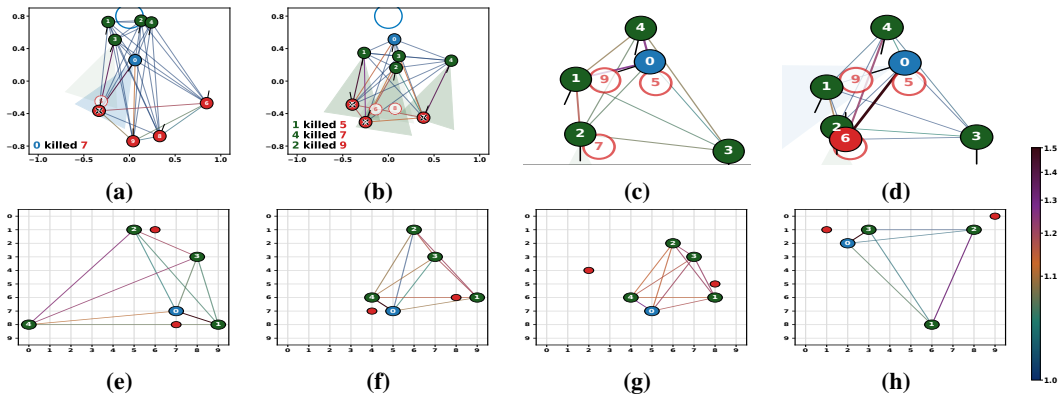



**Figure 4:** Dynamic soft coordination topology in FortAttack (top) and Wolfpack (bottom), team size 5; edge color encodes interaction weight (low → high, see colorbar). Agent 0 (in blue color) denotes *learner-agent*. Agents not shaded in (a, b, c, and d) denote killed agents in the FortAttack domain.

setting, yielding the highest performance WCRP of $0.84$ and indicating substantially more robust zero-shot generalization across environments and team sizes. These generalization results highlight the adaptability of dynamically learned topologies, providing a versatile framework that maintains robust performance across diverse environmental settings, behavioral uncertainties, and team sizes, all without requiring task-specific structural changes.

### 5.2 Interpreting the Soft Coordination Topology

**Relevance-driven, not proximity-driven:** In FortAttack (Figure 4a), right after attacker 5 is eliminated, guards 0 and 3 together immediately focus on attacker 7 to kill. This is also indicated by the high edge-weights of ($\sim$1.4) between (0, 7) and (3, 7). Figure 4(b) illustrates a multi-kill scenario where different guards kill attackers independently. This incident is also clearly reflected in the high edge weights between the corresponding guards and attackers. Figure 4(e) shows an analysis of the Wolfpack domain. Agent 4 is geometrically closer to agent 0 than to 2 or 3. However, instead of teaming up with agents 0 and 1, which are already very close to capturing a prey, agent 4 forms a stronger connection with agents 2 and 3, which are near a second unattended prey. These results show that learned edge weights consistently track task relevance rather than proximity.

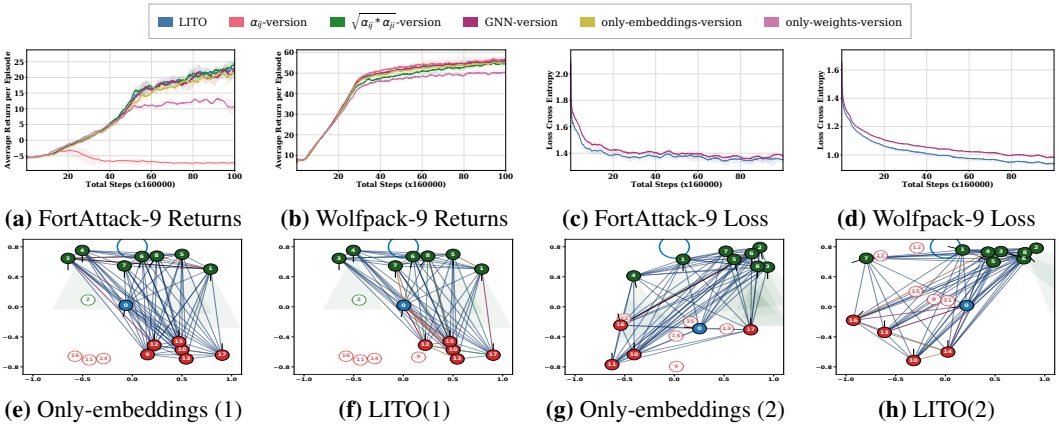
**Event-driven topology:** Figures 4c and 4d are subsequent frames of an instance of the FortAttack domain. In Figure 4d, attacker 6 spawns near a guard cluster, and immediately we see the edge weight of (0-6) jumps to its peak value of $\sim$1.5. Similarly, in the Wolfpack domain, Figure 4f shows agent 0 is paired with agent 4 around a new prey after its earlier pairing with agent 1 has decayed. Subsequently, in the next instant in Figure 4g, we see the edge weight of (0-4) starts to decay the instant that the prey in Figure 4f is captured and a new one respawns elsewhere in Figure 4g. This shows the instant network reconfiguration capability of our proposed method.

**Robustness to *open-environment:*** One of the key challenges in OAHT is that agents can enter and exit at any time. For the FortAttack domain, Figures 4(c and d) capture it, in which agent 6 spawns in the environment. Similarly, for the Wolfpack domain, in Figures 4(g and h), agent 4 exits the environment. In both environments, we observe that, unlike previous methods with fixed topologies that simply reconnect uniformly to the remaining agents, our topology readjusts edge weights toward agents currently relevant to the task as the number of agents in the environment changes.

### 5.3 Ablation Studies

We perform ablation studies on the core design choices of our method by evaluating on both mixed and purely cooperative domains of FortAttack-9 and Wolfpack-9, respectively.

**Edge-weight ablation:** We first examine the edge-weight formula. $\alpha_{ij}$-version, replaces $w_{ij}$ with the raw, one-way attention score $\alpha_{ij}$, while, $\sqrt{\alpha_{ij}\alpha_{ji}}$-version excludes the additive complete-graph prior. As shown in Figure 5(a-b), the two variants fail in complementary ways: $\alpha_{ij}$-version performs




**(a)** FortAttack-9 Returns **(b)** Wolfpack-9 Returns **(c)** FortAttack-9 Loss **(d)** Wolfpack-9 Loss

**(e)** Only-embeddings (1) **(f)** LITO(1) **(g)** Only-embeddings (2) **(h)** LITO(2)

**Figure 5: Ablation Results** Figures (a-d) show learning curves and Loss (CE) curves for the mentioned environments. Figures (e-h) contrast only-embeddings-version ablation with LITO, in FortAttack-9. The edge-weight color mapping is similar to that shown in the interpretation section.

comparably to LITO in Wolfpack but fails to learn in FortAttack, while $\sqrt{\alpha_{ij}\alpha_{ji}}$-version learns in FortAttack, but performs below LITO in Wolfpack.

**Embedding ablation:** Next, we analyze the representation component of our architecture. In Figure 5(a-b), only-embeddings-version removes edge-weights of the coordination graph entirely (i.e., $w_{ij} = 1$), whereas only-weights-version retains the edge-weights but excludes the MAT embeddings, and directly uses the type embeddings from the LSTM. Results show that only-weights-version consistently underperforms, while only-embeddings-version closely follows our full method, highlighting the importance of the contextualized embeddings. Empirically, we observe that the contextualized embeddings are key to overall performance. However, edge weights are vital to the interpretability of the multi-agent interactions in the coordination graph. The attention graphs in Figure 5(e-h) show that edge-weights in our method are noticeably more interpretable and meaningful. In contrast, the weights from only-embeddings-version, computed post-hoc using the same formula, are often uninterpretable and sometimes random. For example, comparing frame 1 in Figure 5(e) (only embedding) shows no meaningful colored edges with the *learner-agent* and arbitrarily high-attention edges among other agents (between 1,3 and 3,12), while in Figure 5(f) (our method) shows considerable attention between the important guards and attackers. The same analysis holds for frame-2 as well in Figure 5(g-h). Therefore, these results show that edge weighting enhances the interpretability of the multi-agent interactions.

**Agent-model ablation:** We also analyze the MAT-based agent-model in our architecture. We replace it with a GNN-based agent model, as in GPL, denoted as GNN-version. This variant performs comparably to our full method in both environments. We attribute this to two factors: a transformer with self-attention and no positional encoding is closely related to a graph attention network (Veličković, 2023), and our agent model is trained with an independent NLL loss that does not fully exploit the richer embeddings the transformer provides. However, we retain the transformer agent model for architectural consistency with the value network and because it yields a consistently lower cross-entropy loss than the GNN variant across both environments Figure 5(c-d), even though the margin is modest and is mostly used qualitatively.

**Conclusion:** We introduced a flexible coordination topology for Open Ad Hoc Teamwork that replaces fixed inductive biases in previous coordination-graph methods with a permutation-invariant, entmax-based attention mechanism. This framework learns important interactions directly from the environment, without assuming the cooperative structures that hinder previous methods against adversaries. In mixed-cooperative setting, our approach effectively handles larger, more complex interactions while remaining competitive with methods tailored to purely cooperative environments. Our detailed analysis shows that the learned topology is both clear and accurate, focusing on relevant interactions and adapting to changes in real time. Additionally, our studies reveal that contextual embeddings is key in performance improvement, while learned edge weights provide valuable clarity on the coordination structure.

## 6 AI DISCLOSURE SECTION

In this work, we have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, implement methods, or interpret results, and generating synthetic data sets, assisting with translation, cleaning and reformatting datasets, and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create and edit software code and to edit the research paper to improve readability. We have reviewed all AI-assisted work. The authors verified all claims made in the paper and proofread the entire manuscript after AI-assisted editing, and AI-assisted code was reviewed and tested by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

## 7 REPRODUCIBILITY STATEMENT

To ensure the reproducibility of the results presented in this paper, we provide the complete source code and configuration files in the supplementary material, including the scripts used for training and evaluation. The proposed method is described in Section 4, and the experimental setup is described in supplementary Section A. Supplementary Section B contains all the details of the architectural specifications and hyperparameter ranges, which together with the provided code allow our results to be reproduced.

## REFERENCES

Noa Agmon and Peter Stone. Leading ad hoc agents in joint action settings with multiple teammates. In *International Conference on Autonomous Agents and Multiagent Systems, AAMAS 2012, Valencia, Spain, June 4-8, 2012 (3 Volumes)*, pp. 341–348. IFAAMAS, 2012. URL http://dl.acm.org/citation.cfm?id=2343625.

Samuel Barrett, Avi Rosenfeld, Sarit Kraus, and Peter Stone. Making friends on the fly: Cooperating with new teammates. *Artificial Intelligence*, 242:132–171, 2017.

Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In *Proceedings of the 37th International Conference on Machine Learning, ICML 2020, 13-18 July 2020, Virtual Event*, volume 119 of *Proceedings of Machine Learning Research*, pp. 980–991. PMLR, 2020.

Lili Chen, Kevin Lu, Aravind Rajeswaran, Kimin Lee, Aditya Grover, Misha Laskin, Pieter Abbeel, Aravind Srinivas, and Igor Mordatch. Decision transformer: Reinforcement learning via sequence modeling. In *Advances in neural information processing systems*, volume 34, pp. 15084–15097, 2021.

Shuo Chen, Ewa Andrejczuk, Zhiguang Cao, and Jie Zhang. Aateam: Achieving the ad hoc teamwork by employing the attention mechanism. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7095–7102, 2020.

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.

Carlos Guestrin, Shobha Venkataraman, and Daphne Koller. Context-specific multiagent coordination and planning with factored mdps. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence and Fourteenth Conference on Innovative Applications of Artificial Intelligence, July 28 - August 1, 2002, Edmonton, Alberta, Canada*, pp. 253–259. AAAI Press / The MIT Press, 2002.

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

Shariq Iqbal and Fei Sha. Actor-attention-critic for multi-agent reinforcement learning. In *International conference on machine learning*, pp. 2961–2970. PMLR, 2019.

Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. *Advances in neural information processing systems*, 30, 2017.

Linghui Meng, Muning Wen, Chenyang Le, Xiyun Li, Dengpeng Xing, Weinan Zhang, Ying Wen, Haifeng Zhang, Jun Wang, Yaodong Yang, et al. Offline pre-trained multi-agent decision transformer. *Machine Intelligence Research*, 20(2):233–248, 2023.

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.

Frans A. Oliehoek, Matthijs T. J. Spaan, and Nikos Vlassis. Optimal and approximate q-value functions for decentralized pomdps. *J. Artif. Intell. Res.*, 32:289–353, 2008. doi: 10.1613/JAIR.2447. URL https://doi.org/10.1613/jair.2447.

Ben Peters, Vlad Niculae, and André F. T. Martins. Sparse sequence-to-sequence models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 1504–1519, Florence, Italy, 2019. Association for Computational Linguistics. doi: 10.18653/v1/P19-1146. URL https://aclanthology.org/P19-1146.

Muhammad A Rahman, Niklas Hopner, Filippos Christianos, and Stefano V Albrecht. Towards open ad hoc teamwork using graph-based policy learning. In *International conference on machine learning*, pp. 8776–8786. PMLR, 2021.

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(178):1–51, 2020.

João G Ribeiro, Cassandro Martinho, Alberto Sardinha, and Francisco S Melo. Assisting unknown teammates in unknown tasks: Ad hoc teamwork under partial observability. *arXiv preprint arXiv:2201.03538*, 2022.

Peter Stone, Gal A. Kaminka, Sarit Kraus, and Jeffrey S. Rosenschein. Ad hoc autonomous agent teams: Collaboration without pre-coordination. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*. AAAI Press, 2010.

Jing Sun, Cong Zhang, and Zhiguang Cao. Collaboration with dynamic open ad hoc team via team state modelling. In *The Twelfth International Conference on Learning Representations (ICLR)*, 2024. URL https://openreview.net/forum?id=BukMU42P3G.

Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z. Leibo, Karl Tuyls, and Thore Graepel. Value-decomposition networks for cooperative multi-agent learning, 2017. URL https://arxiv.org/abs/1706.05296.

Andrea Tacchetti, H. Francis Song, Pedro A. M. Mediano, Vinícius Flores Zambaldi, János Kramár, Neil C. Rabinowitz, Thore Graepel, Matthew M. Botvinick, and Peter W. Battaglia. Relational forward models for multi-agent learning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJlEojAqFm.

Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pp. 330–337, 1993.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Petar Veličković. Everything is connected: Graph neural networks. *Current Opinion in Structural Biology*, 79:102538, 2023.

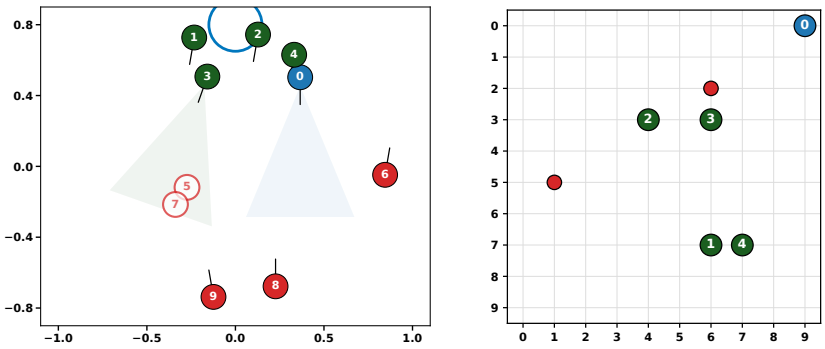
Jianhong Wang, Yang Li, Yuan Zhang, Wei Pan, and Samuel Kaski. Open ad hoc teamwork with cooperative game theory, 2024. URL https://arxiv.org/abs/2402.15259.

Muning Wen, Jakub Grudzien Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. Multi-agent reinforcement learning is a sequence modeling problem. In *Advances in Neural Information Processing Systems*, 2022.

Xinzhi Zhang, Hohei Chan, Deheng Ye, Yi Cai, and Mengchen Zhao. Ad hoc teamwork via offline goal-based decision transformers. In *International Conference on Machine Learning*, 2025.

Xutong Zhao and Yaqi Xie. Multi-level advantage credit assignment for cooperative multi-agent reinforcement learning. In *International Conference on Artificial Intelligence and Statistics, AISTATS 2025, Mai Khao, Thailand, 3-5 May 2025*, volume 258, pp. 2971–2979. PMLR, 2025.

# A Environment Details



**(a)** FortAttack **(b)** Wolfpack

**Figure 6: Environment renderings.** Sample frames from FortAttack (a) and Wolfpack (b). In both, the learner is shown in blue and teammates in green; red denotes attackers in FortAttack and prey in Wolfpack. In FortAttack, each agent's facing direction is marked by a short black line, and an active shooting action is rendered as a pale cone that darkens on a hit.

**Wolfpack.** A purely cooperative predator–prey task on a $10 \times 10$ grid. A pack of hunters must corner a moving prey; a capture occurs when at least two predators are positioned on adjacent sides of the prey. Every hunter in a capturing pack of size $\geq 2$ receives a reward of twice the pack size, while an agent that positions itself next to a prey without a supporting teammate incurs a $-0.5$ penalty. Teammates are drawn from the same nine heuristic/GNN-based predator types (H1–H9) used by GPL (Rahman et al., 2021) and CIAO (Wang et al., 2024), and we use the same number of prey (2) as both. Preys spawn at a different location instantly after being caught. Each episode has a fixed length of 200 steps. Figure 6(b) shows a sample frame: the learner is shown in blue, teammates (predators) in green, and prey in red.

**FortAttack.** A mixed cooperative–competitive task on a continuous 2D plane. Our learner is a guard, alongside other guards, protecting a fort against pretrained adversarial attackers; the learner receives $+3/-3$ for destroying/being destroyed, $\pm 10$ for the team successfully defending/losing the fort, and a $-0.1$ cost per shot. Teammate and opponent behaviours are the pretrained policies as used by GPL. Figure 6(a) shows a sample frame: the learner is shown in blue, guards in green, and attackers in red. Each agent has a facing direction, indicated by a short black line on its body, in which it can shoot; a shooting action is rendered as a pale triangular cone showing the range of the shot, which darkens on a successful hit. An episode terminates when either of the three conditions satisfies: the learner agent is killed by an attacker (note that if other agents are killed, they can respawn during an episode), an attacker manages to break into the fort, or the episode length is reached (200 steps).

**Openness.** Teammates enter and leave stochastically. For Wolfpack, active/waiting durations are sampled uniformly from $[25, 35]$ and $[15, 25]$ timesteps respectively; for FortAttack, an agent's respawn delay is linearly interpolated between 0 and 80 timesteps based on its distance from the agent that eliminated it, following GPL (Rahman et al., 2021).

# B Implementation Details

This section complements Section 4 with details not covered there. The training procedure is summarized in Algorithm 1, and all layer sizes are listed in Table 3.

**Type inference.** As in GPL (Rahman et al., 2021), the joint action-value model and the agent model use separate LSTM type-inference modules, so that their gradients do not interfere. Hidden states of departing agents are removed, and those of newly arriving agents are initialized to zero.

**MAT encoder.** Each MAT encoder consists of two post-norm transformer blocks with `entmax15` (Peters et al., 2019) multi-head attention. Since team sizes differ across the parallel environments

in a batch, the embeddings of each environment are zero-padded to the largest team in the batch before encoding. The attention scores $\alpha_{ij}$ used for the edge weights are taken from the final block and averaged over heads, and $w_{ij}$ is computed for every edge of the complete coordination graph.

**Agent model.** The agent model has its own LSTM and MAT encoder, followed by an MLP head producing $q_\phi(a_t^j \mid s_t)$, and is trained with the negative log-likelihood of the teammates' observed actions.

**Marginalization.** In computing $\bar{Q}_{\pi^i}$, edges incident to the learner contribute $w_{ij} \cdot Q_\delta^{i,j}(a^i, a^j)$ once per teammate, marginalized over $q_\phi(a^j)$, while edges between two teammates are marginalized over both teammates' predicted actions. The teammates' expected individual utilities are also included; they are constant in $a^i$ and therefore do not affect action selection.

**Compute resources.** Experiments were conducted on a workstation equipped with a 96-core AMD Threadripper CPU, 312 GB of RAM, and two NVIDIA RTX 5000 Ada Generation GPUs (32 GB each).

## C Algorithm

**Algorithm 1:** Learning Interaction Topologies for Open ad hoc teamwork (LITO) training procedure

**Input**: parallel environments $E$, total steps $T$, update frequency $U$, soft-update rate $\tau$, loss weight $\lambda_{\text{pred}}$
Initialize the online network (type-inference modules, MAT encoders, utility networks, agent model)
Initialize the target network as a copy of the online network
**for** $t = 1, \dots, T$ **do**
  ▷ Encode the active team and build the soft coordination graph (in all $E$ environments in parallel)
  Build the complete coordination graph over the active team $N_t$
  Encode type embeddings $\{\theta_t^j\}$ via the recurrent type-inference module
  Pad $\{\theta_t^j\}$ to the largest team in the batch; run the MAT encoder to obtain $\{\bar{\theta}_t^j\}$ and attention scores $\{\alpha_{ij}\}$
  Compute edge weights $w_{ij} = 1 + \sqrt{\alpha_{ij}\alpha_{ji}}$
  ▷ Act
  Predict teammate action distributions $q_\phi(a_t^j \mid s_t)$ with the agent model
  Compute individual and pairwise utilities $Q_\beta^j$, $Q_\delta^{j,k}$ from $\{\bar{\theta}_t^j\}$
  Marginalize over $q_\phi$ to obtain $\bar{Q}_{\pi^i}(s_t, \cdot)$
  **if** *exploring (with probability $\epsilon$)* **then**
    Sample $a_t^i$ uniformly at random
  **else**
    $a_t^i \leftarrow \arg\max_{a^i} \bar{Q}_{\pi^i}(s_t, a^i)$
  **end**
  Execute $a_t^i$; observe $r_t$, $s_{t+1}$, the new team $N_{t+1}$, and teammates' actions $a_t^{-i}$
  ▷ Accumulate losses
  Compute the joint action value $Q_\xi^i(s_t, a_t)$ for the observed joint action (Eq. 8)
  Compute the target $y_t = r_t + \gamma \max_{a^i} \bar{Q}_{\pi^i}^{\text{targ}}(s_{t+1}, a^i)$ with the target network
  Accumulate $\mathcal{L}_{\text{TD}} = \big(y_t - Q_\xi^i(s_t, a_t)\big)^2$
  Accumulate $\mathcal{L}_{\text{agent}} = -\log q_\phi(a_t^{-i} \mid s_t)$
  ▷ Update
  **if** $t \bmod U = 0$ **then**
    Take a gradient step on $\mathcal{L}_{\text{TD}} + \lambda_{\text{pred}}\, \mathcal{L}_{\text{agent}}$
    Clear the accumulated losses
    Soft-update the target utility network: $\phi_{\text{targ}} \leftarrow (1-\tau)\phi_{\text{targ}} + \tau\phi$
    Copy the agent-model parameters to the target network
  **end**
**end**

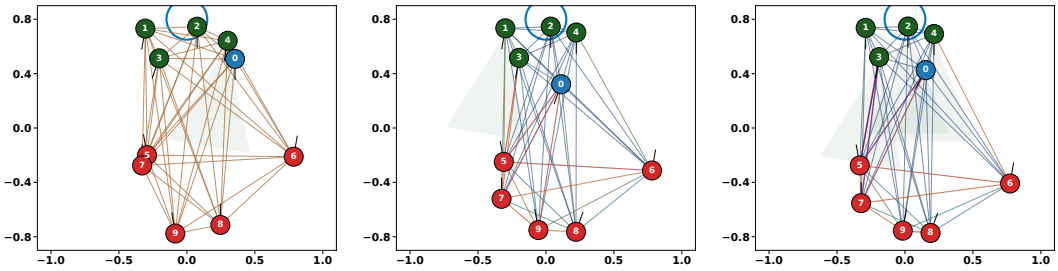
## D Hyperparameters

The training scheme follows GPL (Rahman et al., 2021). Hyperparameters are shared across all methods and environments unless stated otherwise.

**Table 3:** Hyperparameters of LITO. All baselines use the same training hyperparameters, except for the final exploration rate, for which GPL and CIAO retain the value of 0.05 used in their original implementations.

| Hyperparameter | Wolfpack | FortAttack |
|---|---|---|
| *Training* | | |
| Optimizer | Adam | Adam |
| Learning rate | $2.5 \times 10^{-4}$ | $1.0 \times 10^{-4}$ |
| Discount factor $\gamma$ | 0.99 | 0.99 |
| Parallel environments | 16 | 16 |
| Episode length | 200 | 200 |
| Total environment steps | $1.6 \times 10^{7}$ | $1.6 \times 10^{7}$ |
| Update frequency | every 4 steps | every 4 steps |
| Target network update | Polyak, $\tau = 10^{-3}$ | Polyak, $\tau = 10^{-3}$ |
| Gradient-norm clipping | 5.0 | 5.0 |
| Agent-model loss weight $\lambda_{\text{pred}}$ | 1.0 | 1.0 |
| $\epsilon$ decay | linear from 1.0 over 2500 episodes | linear from 1.0 over 2500 episodes |
| Final exploration rate $\epsilon_{\min}$ | 0.01 (LITO); 0.05 (GPL, CIAO) | 0.01 (LITO); 0.05 (GPL, CIAO) |
| Evaluation | 5 episodes every $1.6 \times 10^{5}$ steps | 5 episodes every $1.6 \times 10^{5}$ steps |
| Random seeds | 6 | 6 |
| *Architecture* | | |
| Number of actions $\lvert A \rvert$ | 5 | 8 |
| Type-inference encoder | FC(100)→FC(100)→LSTM(100)→FC(70) | FC(64)→FC(64)→LSTM(128)→FC(128) |
| Embedding dimension | 70 | 128 |
| MAT encoder (value network) | 2 blocks, 2 heads, FF 256 | 2 blocks, 2 heads, FF 256 |
| Individual utility MLP | FC(60)→FC(60)→FC($\lvert A \rvert$) | FC(128)→FC(128)→FC($\lvert A \rvert$) |
| Pairwise utility MLP $f_\delta$ | FC(60)→FC(60)→FC($\lvert A \rvert \cdot r$) | FC(128)→FC(128)→FC($\lvert A \rvert \cdot r$) |
| Pairwise rank $r$ | 5 | 6 |
| Agent-model encoder | FC(100)→FC(100)→LSTM(100)→FC(70) | FC(64)→FC(64)→LSTM(128)→FC(128) |
| MAT encoder (agent model) | 2 blocks, 2 heads, FF 30 | 2 blocks, 2 heads, FF 40 |
| Agent-model head | FC(20)→FC($\lvert A \rvert$) | FC(128)→FC($\lvert A \rvert$) |
| Activation | ReLU | ReLU |

## E EMERGENCE OF THE LEARNED TOPOLOGY



**(a)** Initial Frame **(b)** Mid-training Frame **(c)** End of training Frame

**Figure 7: Emergence of the learned coordination topology by LITO in FortAttack-9**. Edge colour encodes $w_{ij}$ (colour bar as in Figure 4); the learner is shown in blue, guards in green, and attackers in red.

Figure 7 shows the coordination topology learned by LITO in FortAttack-9 at three points during training, rendered for the same scenario to allow a direct comparison. At initialization, the edge weights $w_{ij}$ show no clear structure. During mid-of training (around 25%), the graph already concentrates on the correct interactions, linking guards to the attackers near them, but the weights on these edges remain low, indicating that the model has identified the relevant interactions without yet being confident in them. By the end of training, the same sparse structure is retained with sharply higher weights: the strongest edges link each guard to the attacker it is engaging, while edges to distant or inactive agents are suppressed. Together with the analysis in Section 5.2, this indicates that the learned topology is non-trivial: its structure is absent at initialization and is acquired through training, and the interactions it emphasizes correspond to meaningful relationships in the task rather than to a fixed or uniform pattern.'

## F RAW SCORES FOR TEAM GENERALIZATION EXPERIMENT

**Table 4: Zero-shot team size generalization (raw returns), models trained on maximum size 3, evaluated on sizes 5 and 9.** Bold denotes the best raw mean return per row; note that in FortAttack, our method's mean return falls within one standard deviation of GPL's, indicating the two are statistically comparable despite GPL's nominal lead (see normalized comparison in Table 2).

| Env. | Size | GPL | CIAO-C | CIAO-S | LITO | Open-OTAF |
|---|---|---|---|---|---|---|
| Fort. | 5 | $\mathbf{23.63 \pm 7.58}$ | $0.22 \pm 1.05$ | $-0.66 \pm 3.58$ | $20.63 \pm 7.53$ | $-8.33 \pm 4.31$ |
| Fort. | 9 | $\mathbf{23.34 \pm 6.48}$ | $-2.29 \pm 1.21$ | $-0.90 \pm 3.35$ | $19.52 \pm 6.29$ | $-5.92 \pm 6.77$ |
| Wolf. | 5 | $37.57 \pm 7.20$ | $45.51 \pm 1.50$ | $\mathbf{55.66 \pm 1.83}$ | $48.65 \pm 1.40$ | $39.32 \pm 2.26$ |
| Wolf. | 9 | $33.50 \pm 9.21$ | $48.69 \pm 0.46$ | $\mathbf{51.86 \pm 1.91}$ | $43.29 \pm 1.80$ | $36.99 \pm 2.38$ |

In FortAttack, our model successfully scales without catastrophic forgetting, achieving competitive returns (20.63 for size 5; 19.52 for size 9) and maintaining parity with GPL, while the CIAO baselines struggle to generalize in this mixed environment. In Wolfpack, our method achieves returns of 48.65 (size 5) and 43.29 (size 9). While CIAO-S retains the highest absolute performance here, benefiting from a predefined sparse graph that scales efficiently in this purely cooperative setting, our dynamic topology demonstrates vastly superior scalability compared to the GPL baseline (37.57 and 33.50). As team size increases, GPL's dense message-passing becomes highly susceptible to noise accumulation; our attention mechanism successfully mitigates this by dynamically filtering less relevant agents.