MOHITO: Multi-Agent Reinforcement Learning using Hypergraphs for Task-Open Systems

Gayathri Anil[1] **Prashant Doshi**[1] **Daniel Redder**[1] **Adam Eck**[2] **Leen-Kiat Soh**[3]

[1]THINC Lab, School of Computing, University of Georgia, Athens, GA, USA
[2]Computer Science Department, Oberlin College, Oberlin, Ohio, USA
[3]School of Computing, University of Nebraska, Lincoln, Nebraska, USA

Abstract

Open agent systems are prevalent in the real world, where the sets of agents and tasks change over time. In this paper, we focus on task-open multi-agent systems, exemplified by applications such as ridesharing, where passengers (tasks) appear spontaneously over time and disappear if not attended to promptly. Task-open settings challenge us with an action space which changes dynamically. This renders existing reinforcement learning (RL) methods–intended for fixed state and action spaces–inapplicable. Whereas multi-task learning approaches learn policies generalized to multiple known and related tasks, they struggle to adapt to previously unseen tasks. Conversely, lifelong learning adapts to new tasks over time, but generally assumes that tasks come sequentially from a static and known distribution rather than simultaneously and unpredictably. We introduce a novel category of RL for addressing task openness, modeled using a task-open Markov game. Our approach, MOHITO, is a multi-agent actor-critic schema which represents knowledge about the relationships between agents and changing tasks and actions as dynamically evolving 3-uniform hypergraphs. As popular multi-agent RL testbeds do not exhibit task openness, we evaluate MOHITO on two realistic and naturally task-open domains to establish its efficacy and provide a benchmark for future work in this setting.

1 INTRODUCTION

In multi-agent systems, each decision-making agent must determine a strategy to achieve collective and/or individual objectives. Learning how to coexist and effectively operate within a shared environment with other agents is challenging but well-studied in *closed* environments where the tasks being accomplished are fixed in time and known in advance. More challenging is learning how to act in environments with *task openness*: a phenomenon where the set of objectives or tasks is neither static nor predefined.

Consider ridesharing, such as Uber Pool, operated by autonomous driver agents or robotaxis. A task in ridesharing is to transport a passenger to their destination. However, passengers enter spontaneously, in unbounded quantity, with a frequency influenced by exogenous factors. Passengers can also withdraw causing tasks to disappear from the system. Such task openness makes the set of action choices transient, alters action-dependent elements of the problem such as the reward function, and may introduce new relationships between agents. As such, actions optimal for one state under one set of tasks might be different from those in the same state under a different set of tasks. Hence, an agent must reason about the changing meaning of actions and availability thereof caused by the entry and exit of tasks.

Recent generalizations of multi-agent RL (MARL) targeting related challenges have produced three important categories of methods. First, *multi-task* learning algorithms learn from multiple related tasks (e.g., Tanaka and Yamamura 2003, Omidshafiei et al. 2017, Zhang et al. 2023) but falter when novel tasks appear. Second, *lifelong* learning enables ongoing adaptation to new tasks over time (e.g., Thrun and Mitchell 1995, Chen and Liu 2018, Skrynnik et al. 2024) but assumes a sequential, and not simultaneous, arrival of the tasks from a known distribution. Third, out-of-distribution learning enables agents to detect when their current tasks are different from training (e.g., Sedlmeier et al. 2020, Haider et al. 2023) but does not say how agents should use that information. In short, extant MARL may not apply under task openness due to the underlying constraint where policies map states to a static action set.

In the context of the novel challenges brought about by task openness to MARL, this paper contributes the following:

- A first and general decision-making model, *task-open*

Markov games (TaO-MG), for such multi-agent settings.

- A hypergraph-based knowledge representation schema modeling the relationships between agents, tasks, and actions amid openness.
- MOHITO, a deep reinforcement learning method for TaO-MG. MOHITO interprets the hypergraphs via a graph neural network to implement its multi-agent actor-critic schema, thereby learning a relative evaluation of available actions.
- We present two domains that naturally manifest this form of openness: rideshare [Eck et al., 2023] and wildfire suppression [Eck et al., 2020] because extant MARL testbeds generally do not exhibit task openness.

Evaluations on these domains using standard (domain-agnostic) metrics and domain-centric ones establish MOHITO's efficacy and offer insights into its behavior. Importantly, this is the first MARL method to fully target task openness, thereby stimulating further development for this new and pragmatic multi-agent decision-making setting.

2 BACKGROUND

We situate task openness in the broader context of open agent systems followed by a review of a well-known framework for modeling multi-agent interactions.

2.1 OPEN AGENT SYSTEMS

Openness can manifest in various ways. A recent survey [Eck et al., 2023] as well as prior discussions [Shehory, 2000, Calmet et al., 2004, Jumadinova et al., 2014] recognize three types: *agent openness*, where the set of agents acting in the environment changes over time; *task openness*, where the set of tasks changes over time; and *frame/type openness*, where agents' frames (capabilities, preferences, and reasoning processes) can change, such as when agents acquire new abilities or change roles. Among these categories of open systems, agent openness has received the most attention whereas task openness remains understudied.

Agent openness was defined more than two decades ago as adding agents beyond the initial number present in a system [Shehory, 2000]. Subsequent work defines the degree of openness as the complexity of the minimal transformation of a system to add or remove an agent [Jamroga et al., 2013]. Since then, reasoning methods have shown that explicitly predicting the presence or absence of other agents, where possible, improves global system behavior [Chandrasekaran et al., 2016, Cohen et al., 2017, Eck et al., 2020, Kakarlapudi et al., 2022]. This leads to the thought that explicitly modeling task openness may have similar success. However, new tasks often arrive due to exogenous factors (e.g., end of a large gathering resulting in many ride-hailing passengers), which may not be possible to model.

2.2 MARKOV GAMES

Markov games are stochastic games that commonly formalize the multi-agent decision-making process of learning agents that optimize their individual cumulative rewards in competitive (self-interested) or cooperative environments [Shapley, 1953, Littman, 1994]. Formally, a Markov game is represented by:

$$M = \langle Ag, S, A, T, R, \gamma, s_0 \rangle$$

where • Ag is the set of agents operating in the environment; • S is the set of states of the environment encapsulating different situations agents can face; • $A = \Pi_{i \in Ag} A_i$ is the joint action set giving possible combinations of actions taken simultaneously by all agents, with A_i the action set of agent i; • $T : S \times A \times S \to [0, 1]$ is the transition function, specifying the probability of the problem state transitioning from state s to s' when the agents perform their joint action $\mathbf{a}$; • $R : S \times A \to \mathbb{R}^{|Ag|}$ is the reward function, specifying the collection of individual rewards earned by each agent on joint action $\mathbf{a}$ in state s; • $\gamma \in (0, 1)$ represents the discount factor for uncertain future rewards; and • $s_0 \in S$ denotes the initial state of the environment.

In a RL context, the objective of each agent $i \in Ag$ in a Markov game is to learn a policy $\pi_i : S \to A_i$ that prescribes actions that maximize the agent's sum of discounted cumulative rewards:

$$\mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R_i(s_t, \mathbf{a}_t | s = s_0) \right]$$

where $R_i(s_t, \mathbf{a}_t)$ is the reward of agent i when i chooses action $\pi_i(s_t)$ in state s_t according to its learned policy and the other agents choose the remaining actions in $\mathbf{a}_t$.

3 TASK-OPEN MAS

Many real-world environments do not abide by the assumptions under which RL generally operates; that the set of tasks agents seek to complete is known in advance, and those tasks remain in the environment until completed.

Instead, for example, drivers in a ridesharing application (e.g., Uber, Lyft) with vacancy in their vehicles must decide whether to accept new passengers that arrive unexpectedly in its environment. For larger vehicles (e.g., Uber Pool), the driver must balance the needs of multiple simultaneous tasks like multiple passengers sharing the same vehicle but desiring different destinations while possibly accommodating new passengers.

As such, new tasks and goals are introduced over time, and their presence may alter previously planned or learned behavior (e.g., picking up a new passenger changing the

intended dropoff order of existing passengers) or cause such behavior to not be optimal.

Tasks can be seen as outcomes of exogenous or endogenous events which cause tasks to enter or exit. Exogenous factors are driven by nature and independent of the agents' actions, whereas endogenous ones can be influenced by agents' actions.

For a more nuanced view, let $X = \langle \mathcal{T}^{\tau}, \mathcal{T}^{\omega} \rangle$, where $\mathcal{T}^{\tau}$ includes parameters of tasks resulting from exogenous events such as those causing new passengers to arrive, and $\mathcal{T}^{\omega}$ contains parameters of tasks affected by endogenous events such as soliciting a passenger. In some circumstances, tasks in $\mathcal{T}^{\tau}$ may be predicted from experience, although reactive behavior is likely necessary.

3.1 DYNAMIC RIDESHARING

We illustrate task openness using the domain of ridesharing, labeled as Rideshare, in Fig. 1. Each driver agent i can transport up to p_i passengers simultaneously. Each passenger (task) has a pickup location, a destination, and a fare. Each agent receives the following information from the environment: its own position and capacity, the location of other agents, and the position, destination, and fare of its accepted passengers and those awaiting service.

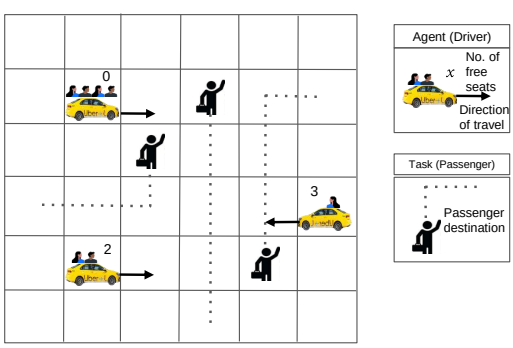


Figure 1: A rideshare driver operates a vehicle in a task-open MAS where new tasks (passengers) suddenly appear and existing tasks suddenly exit leading to *open* action sets A_i.

Because passengers appear dynamically, the set of actions A_i of a driver agent i changes over time. For each unserved passenger, the agent can choose to *accept* the passenger for transport. For each accepted passenger who is not in their assigned car, the agent can choose to drive to their *pickup* location. Finally, for each riding passenger, the agent may drive to their destination for *drop off*. New passengers increase the size of A_i by adding these actions, while completed tasks reduce it. The rewards, R, also change as the composition of tasks in the environment changes. New passengers introduce new fares that provide new opportunities for agents to earn rewards. As a result, each agent's learned utility function must adapt to changes in the set of passengers caused by task openness.

Note that changes in the set of passengers over time *cannot simply be modeled as agent openness*, for which there are existing reinforcement learning and planning solutions [Cohen et al., 2017, Eck et al., 2020, Rahman et al., 2021, Kakarlapudi et al., 2022]. This is because a passenger is not an autonomous actor in Rideshare; only driver agents deliberately choose actions to complete tasks.

4 TASK-OPEN MARKOV GAMES

Task openness complicates agent decision making within a Markov game by causing many components of the game to potentially change according to task dynamics. Specifically, the states reflect the features of each task. Changes in task features alter the factored state variables related to the environment (e.g., the current location of each passenger). Tasks entering or leaving change the space of unique actions available to agents. The transition function must handle the dynamics of both these spaces and possible stochastic changes with current tasks. The reward function also changes under different tasks, consequently changing the agent's ultimate objective as the expected value of an action may change dramatically as new actions become available.

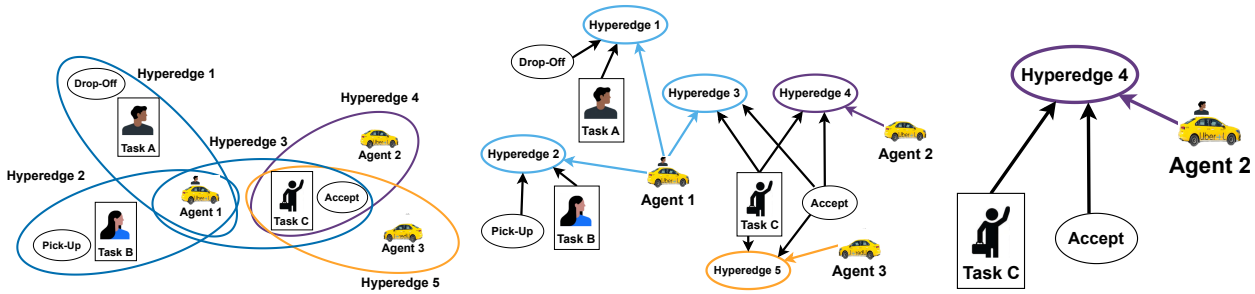
To support agent reasoning about task openness and handle its compound impacts on the viability and optimality of an action, we propose a time-varying model called the **task-open Markov game** (TaO-MG), formalized as:

$$\text{TaO-MG} \triangleq \langle M, X, \Psi \rangle$$

where M is the *current* base decision-making model of the problem instance. Here, it is a Markov game as previously described in Section 2.2. X includes (features of) the *current* set of tasks in the system. Ψ is the generator function that transforms M when exogenous or endogenous events change the set of available tasks at a given step. These tasks populate $X' = \langle \mathcal{T}'^{\tau}, \mathcal{T}'^{\omega} \rangle$. Given X', Ψ updates the current decision-making model M to M',

$$M' = \begin{cases} \Psi(M, X') & \text{if } X \text{ transitions to } X' \\ M & \text{otherwise} \end{cases}$$

The components of Ψ operate on parameters of the base model M and generate new ones. Specifically, $\Psi_S : S \times X' \to S'$ updates the state space to include representations of newly added tasks and remove those of exited tasks. $\Psi_A : A \times X' \to A'$ updates the action space, combining existing actions with those required for new tasks and removing actions no longer associated with any present tasks. $\Psi_T : T \times X' \to T'$, adapts the transition function to incorporate the new states and actions while excluding those related to exited tasks. $\Psi_R : R \times X' \to R'$ similarly updates the



(a) (b) (c)

Figure 2: (a) An interaction hypergraph representation of agents, tasks, and action spaces in the TaO-MG model for a simple instance of Rideshare. (b) The corresponding 2D, critic incidence graph representation of the hypergraph. (c) Agent 2's observation graph: its observed tasks and available hyperedges.

reward function to account for any new or removed states and actions. Note that the rest of the parameters of the Markov game do not change.

Let $\tau = \langle s_t, \mathbf{a}_t, X_t \rangle$ denote a sample experience at timestep t. The objective of each agent $i \in Ag$ in TaO-MG is to learn a policy $\pi_i : S \times X \rightarrow A_i$ which maximizes the agent's expected sum of discounted rewards, $\mathbb{E}_{\tau\sim\pi}\left[\sum_{t=0}^{\infty} \gamma^t R'_i(s_t, \mathbf{a}_t | s = s_0)\right]$, in the task-open context where the reward function may change, $R' = \Psi_R(R, X)$.

We utilize a Markov game representation for a given X. We assume that the environment is fully observable and that rewards can be individual. This base model may be replaced by a different type of decision-making process as appropriate (e.g., multi-agent MDP [Boutilier, 1996], interactive partially observable Markov decision process (I-POMDP) [Gmytrasiewicz and Doshi, 2005], or decentralized (Dec-POMDP) [Oliehoek and Amato, 2016]). A related model to TaO-MG is the time-varying Markov decision process (TVMDP) [Liu and Sukhatme, 2018, Ornik and Topcu, 2021]. TVMDP models a single agent's decision process in environments with exogenous changes to the transition function. Our TaO-MG model can be viewed as extending TVMDPs to multi-agent settings with *dynamic* states, actions, and rewards.

5 MARL UNDER TASK OPENNESS

We present the first RL method for TaO-MG, called Models Of Hyper Interactions under Task Openness (MOHITO), that adopts a graphical problem representation and engages in actor-critic based model-free learning.

5.1 INTERACTION HYPERGRAPHS

The primary challenge in designing a learning algorithm for a task-open setting lies in accommodating the changing set of actions from the openness. Conventional RL algorithms learn $\pi : S \rightarrow A$ by evaluating each state-action pair. In contrast, our work performs a relative comparison of the *current set of actions* to choose an action. That is, a generalized policy must also consider the set of tasks and the corresponding actions available to the agent within that state. It is not trivial to represent these dynamic relationships induced by task openness. To address this, we represent actions, tasks, and agents through a graphical construct, which we call an *interaction hypergraph*. This hypergraph generalizes coordination graphs [Guestrin et al., 2002, Boehmer et al., 2020], which limit interactions to between agents only.

Definition 1 (Interaction Hypergraph) *An* ***interaction hypergraph*** *is a 3-uniform (tripartite) hypergraph* $\mathcal{G} = \langle \mathcal{N}, \mathcal{X}, \mathcal{A}, E \rangle$ *comprised of three types of nodes:* agent nodes $\mathcal{N} = \{node(i) \mid i \in Ag\}$ *store the state-specific information of agents* Ag*, notably including observations of other agents;* task nodes $\mathcal{X} = \{node(x) \mid x \in X\}$ *contain state information about the current tasks* X *in the environment;* action nodes $\mathcal{A} = \{node(a) \mid a \in \bigcup_{i\in Ag} A_i\}$ *hold details about the unique actions currently available to the agents.* E *is a set of 3-uniform hyperedges where each hyperedge (agent, task, action) contains one node from each set* $\mathcal{N}$*,* $\mathcal{X}$*, and* $\mathcal{A}$*, respectively.*

We present the complexity of encoding knowledge of the environment in hypergraphs in Lemma 1, with its derivation in Appendices A.1 and A.2. Our bound is quadratic, but the more common case of tasks that share all unique actions has a linear bound.

Lemma 1 *[Interaction graph complexity] The number of hyperedges is bounded by* $\mathcal{O}(|\mathcal{N}||\mathcal{X}||\mathcal{A}|)$*, and the number of action nodes is bounded by* $\mathcal{O}(|Ag||A_i|)$*. Thus, we can construct the graph in* $\mathcal{O}(|Ag|^2|X||A_i|)$ *space and time.*

Figure 2(a) illustrates an interaction hypergraph for a setting of Rideshare, introduced previously in Sec. 3.1. We

mitigate the representational complexity of working with hypergraphs by transforming them into 2-uniform bipartite graphs, referred to as Levi incidence graphs. Figure 2(b) shows the 2-uniform bipartite graph for the tripartite hypergraph in Fig. 2(a). This regular graph adds an additional set of nodes E which denote hyperedges. We include *InteractionGraph*, a polynomial-time algorithm in Appendix A.2, for constructing an interaction hypergraph from agent observation, and empirical time and memory profiling of our proposed approach in Appendix A.3.

5.2 MOHITO

MOHITO leverages the interaction hypergraph's explicit representation of task-open problems to learn a centralized training decentralized execution (CTDE) actor-critic model in a task-open environment. Each agent has a local actor (or policy) and a local critic. Agents' policies are learned in a centralized manner by exchanging local observations among the critics, thereby leveraging joint observations of the state and the joint action set. This architecture, analogous to MADDPG's [Lowe et al., 2017], allows each agent to learn considering both individual and collective preferences yet remains an autonomous agent during execution.

As illustrated in Fig. 3, both the actor and critic are graph neural networks (GNN), specifically graph attention networks [Veličković et al., 2018], which are representations amenable to dynamic environments. We use Gilmer et al. [2017]'s message passing framework to facilitate aggregating knowledge of the current state of agents, tasks, and available actions. Knowledge is aggregated into the hyperedge nodes of the Levi incidence graph. Hence, the agents learn how to combine information about dynamic task and action sets as represented in the interaction hypergraphs, so that they can reason about their task-open environment.

The actor network processes the agent's *observed interaction incidence graph*, or *observation graph*, $\mathbf{G_O}=\{G_1, ..., G_{|Ag|}\}$ (e.g., Fig. 2(c)). The updated features of each hyperedge node $ed_{ik} \in E$ for the ith agent and k^{th} node are the actor's evaluation of that hyperedge's connected (agent, task, action) nodes. At each decision point, agent i's actor GNN deterministically selects the hyperedge with the highest evaluation $ed_i = \text{argmax}_k \sum_f ed_{ik}$ where f are the features of ed. The agent then performs the action linked to that hyperedge. Each actor $\pi^{\boldsymbol{\theta}_i}$ is a deterministic model with target $\pi^{\boldsymbol{\theta}'_i}$. Actor i's loss function,

$$L_{\pi_i} = \frac{1}{B}\left[\sum_{j \in \text{batch}} -Q_i^{\boldsymbol{\phi}}(G^j_{\mathbf{C}}, ed^j)\right] + \lambda_A|\boldsymbol{\theta}_i - \boldsymbol{\theta}'_i|, \quad (1)$$

includes the ith critic's evaluation Q_i over the agent's action preferences, the hyperedges from all agents ed^j, and the critic graph $G^j_{\mathbf{C}}$, over a batch of samples $j \in$ batch, B is the size of the batch, and a regularization component moderated by $\lambda_A \geq 0$. The policy gradient is computed as,

$$\nabla L_{\pi_i} = \frac{1}{B}\left[\sum_{j \in \text{batch}} \nabla_{\boldsymbol{\theta_i}} \pi^{\theta_i}(G^j_i) \nabla_{ed^j_i} - Q_i^{\boldsymbol{\phi}}(G^j_{\mathbf{C}}, ed^j)\right] + \lambda_A \frac{\boldsymbol{\theta_i} - \boldsymbol{\theta'_i}}{|\boldsymbol{\theta_i} - \boldsymbol{\theta'_i}|}. \quad (2)$$

The critic network processes the *critic interaction incidence graph* or *critic graph* (e.g., Fig. 2(b)), which combines agent observation graphs and extracts all observed task features by performing a disjunctive graph join over all observation graphs, $G_{\mathbf{C}} = \bigvee_{i=1}^{|Ag|} G_i$ [Bergami et al., 2016]. Agent i's critic value $Q_i^{\boldsymbol{\phi}}$ with target network $Q_i^{\boldsymbol{\phi}'}$ takes as input the critic graph, $G_{\mathbf{C}}$, and all ed_i. The target networks θ' and ϕ' are slowly updated, every K batches, by the difference between the main and the target weighted with hyperparameters ψ_A and ψ_Q, respectively. The critic loss,

$$L_{Q_i} = \frac{1}{B}\sum_{j \in \text{batch}} \left(r^j_i + \gamma Q_i^{\boldsymbol{\phi}'}(G'^j_{\mathbf{C}}, ed'^j) - Q_i^{\boldsymbol{\phi}}(G^j_{\mathbf{C}}, ed^j)\right)^2 + \lambda_C|\boldsymbol{\phi} - \boldsymbol{\phi}'| \quad (3)$$

is the mean squared error between expected and calculated Q values for each critic with a similar $\lambda_C \geq 0$ regularization parameter. The gradient remains the same as MADDPG's critic gradient except for the use of ed over a. The architectures, hyperparameters, and operational details of the actor and critic networks are discussed further in Appendix A.4.

Algorithm 1 presents the algorithm for MOHITO. After collecting observation graphs, which capture the varying sets of tasks and associated actions, into a batch (lines 4-9), the algorithm generates the critic graphs (line 10) and utilizes the batch to engage in actor-critic training (lines 12-20) with the loss functions defined previously in Eqs. 1 and 3. We show the complexity of MOHITO in Theorem 1 with its analysis in Appendices A.1, and A.2. Sizes $|\theta|$ and $|\phi|$ equal to #layers $\cdot$ #heads $\cdot f^2 +$ #heads $\cdot f$ where f is the feature size of $\mathcal{N}$ [Veličković et al., 2018]. Number of heads and layers are structural parameters of a GAT.

Theorem 1 *[MOHITO complexity] We perform $\mathcal{O}(|Ag|)$ network queries in one iteration of training. Each query is bounded by $\mathcal{O}(|\theta||G_{\mathbf{C}}|)$. All other procedures are dominated by one query. One iteration of MOHITO for $|batch| = B$ is bounded by $\mathcal{O}(B|\theta||Ag|^3|X||A_i|)$.*

6 EXPERIMENTS

To evaluate the effectiveness of MOHITO, we empirically test its performance in instances of two experimental domains: the aforementioned Rideshare and the Wildfire Suppression benchmark for studying open agent systems [Chandrasekaran et al., 2016, Eck et al., 2020, Kakarlapudi

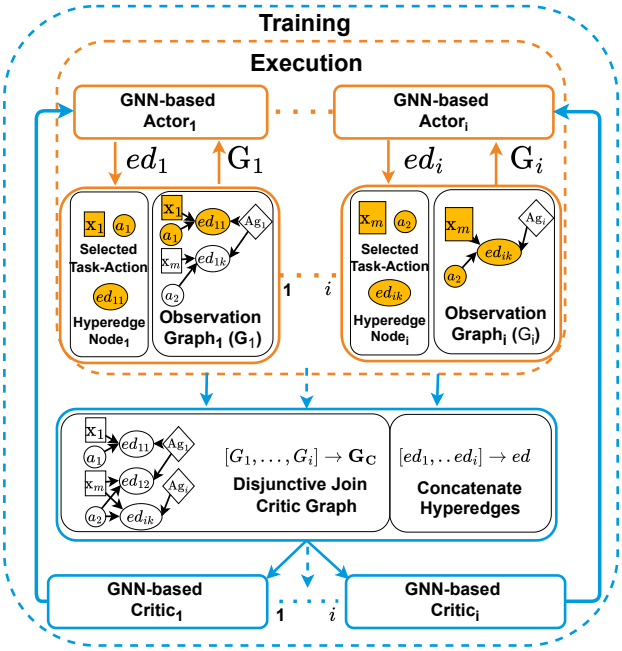


Figure 3: MOHITO uses an actor-critic schema with one actor and critic per agent. Observations (as incidence graphs) are shared between critics enabling centralized training.

Algorithm 1 MOHITO

for $episode \leftarrow 1$ to N **do**
  Get obs from environment with current tasks X
  $\mathbf{G_O}$:$[G_1, ..., G_{|Ag|}] \leftarrow$ InteractionGraph(obs)
  while $episode$ not complete **do** ▷ Online
    $a_i, ed_i \leftarrow \pi^{\boldsymbol{\theta}_i}(G_i) \quad \forall i \in Ag$
    $\mathbf{a} \leftarrow \langle a_1, ..., a_i \rangle$ or ϵ-greedy
    $obs', r \leftarrow$ environment($\mathbf{a}$)
    $\mathbf{G'_O} \leftarrow$ InteractionGraph(obs')
    $a'_i, ed'_i \leftarrow \pi^{\boldsymbol{\theta}'_i}(G'_i) \quad \forall i \in Ag$
    $G_\mathbf{C} \leftarrow \left(\bigvee_{i=1}^{|Ag|} G_i\right), \quad G'_\mathbf{C} \leftarrow \left(\bigvee_{i=1}^{|Ag|} G'_i\right)$
    batch $\leftarrow$ batch $\cup$ $(G_\mathbf{C}, \mathbf{ed}, r, G'_\mathbf{C}, \mathbf{ed}')$
    if $|$batch$| = B$ **then** ▷ Offline learning
      Obtain L_{Q_i} from Eq. 3 $\forall i \in Ag$
      Backprop. L_{Q_i} and update $Q^{\phi_i} \quad \forall i \in Ag$
      Obtain L_{π_i} from Eq. 1 $\forall i \in Ag$
      Backprop. L_{π_i} and update $\pi^{\boldsymbol{\theta}_i} \quad \forall i \in Ag$
      batch $\leftarrow \emptyset$
  for agent i after every K episodes **do**
    $\boldsymbol{\theta}'_i \leftarrow \psi_A \times \boldsymbol{\theta}_i + (1 - \psi_A) \times \boldsymbol{\theta}'_i$
    $\boldsymbol{\phi}'_i \leftarrow \psi_Q \times \boldsymbol{\phi}_i + (1 - \boldsymbol{\psi}_Q) \times \boldsymbol{\phi}'_i$

et al., 2022]. Domain implementations and MOHITO's code are available at `https://github.com/oasys-mas/mohito-public`.

6.1 DYNAMIC RIDESHARING

We consider settings with $|Ag|$ = 2, 3, and 4 driver agents, as well as 3 levels of increasing task openness. These nine environments start with a set of initial tasks $|X_0| \sim [|Ag| - 1, |Ag| + 3]$. Due to task openness, additional tasks are added stochastically over time based on the **openness level (OL)**, our simulation of the entry of exogenous tasks. Levels 1, 2 and 3 add 6, 9, and 12 ride requests throughout the episode at randomly sampled times, respectively. At each of those times, 1-3 new tasks are introduced with probabilities 70%, 20%, and 10%, respectively. We formally define this domain as a TaO-MG in Appendix A.5.

MOHITO's training in each configuration of number of agents and openness level generally spans 20,000 episodes by when we observed stability in the loss. Each episode consists of at most 100 steps or until all tasks are completed (scheduled passengers served). We checkpoint each model every 50 model updates (every batch-size steps, see Appendix A.4 for parameters) and utilize the best performing policy from checkpoints. We evaluate Rideshare instances on 10 episodes for each set of agents and openness level.

6.2 DYNAMIC WILDFIRE SUPPRESSION

We use Wildfire Suppression as our second domain, where agents coordinate to extinguish fires before they burn out. Agents do not move. They choose to *suppress* a fire they can reach, or engage in *No-Op* to refill their suppressant. An agent without suppressant may only *No-Op*; agents thus leave the environment when out of suppressant. A fire's size (small or medium) dictates how many agents must attack the fire to stochastically decrease its intensity. In this cooperative domain, rewards are joint and conditioned on the size of fires. Fires burn out with a penalty of -10 or -25 and put-out fires award +20 or +400 for small and medium fires, respectively. Medium fires are more dangerous and require collaboration, so we set a higher reward for their put outs and a higher penalty for their burn outs. Fires can increase in intensity over time if not attended to, and a fire spreads to adjacent cells using a realistic wildfire spread model [Boychuk et al., 2009, Ure et al., 2015].

Figure 4: Wildfire Suppression starting states. Fires start at intensity 2 or 3 (4 is burned out and 0 is put out). Fires lit in cells with a caution icon start at intensity 3.

This environment is naturally task-open. Each grid-cell is a large sector and therefore allows multiple fires in a sector (we cap at 13). Some cells have more fuel, and fires lit in those cells start at a high intensity (of 3), which is one intensity level away from burning out. We formally define this problem domain as a TaO-MG in Appendix A.5. MOHITO's training in each configuration again spans 20K episodes, and each episode consists of at most 100 steps or until all fires are put out or burn out. We evaluate Wildfire Suppression instances on 20 episodes for each start state shown in Fig. 4.

6.3 BASELINES AND MEASURES

As there are no prior MARL solutions for task-open environments, we compare MOHITO with effective baselines inspired by domain-specific strategies.

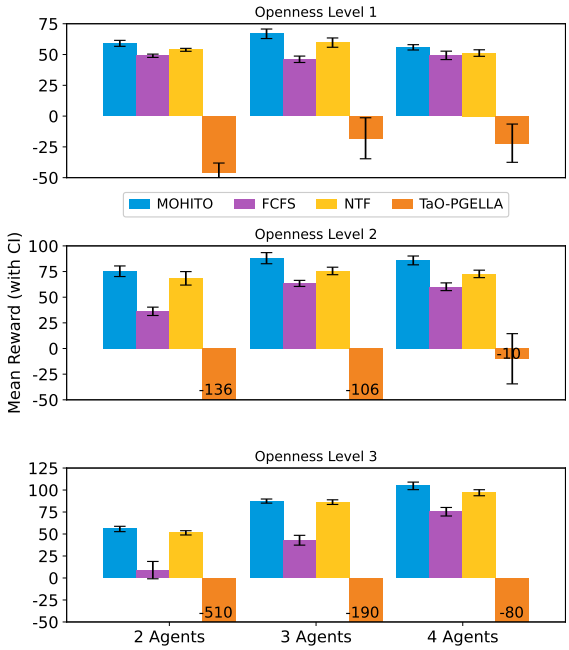
- *FCFS*, a first-come, first-serve policy prioritizing the longest waiting task (as in Uber rides from airports [Uber, 2025], offensive firefighting [U.S. Fire Admin, 2018]).
- *NTF*: a nearest-task first greedy policy prioritizing the task nearest to completion (e.g., defensive firefighting U.S. Fire Admin [2018]).
- *MOHITO-NoTaskNodes*: an ablation of MOHITO where we do not include task nodes in the incidence graph. Actors choose actions without regard to which task they belong to. The task is then selected randomly among the tasks with that action.
- TAO-PGELLA: a multi-agent, task-open adaptation of PG-ELLA [Ammar et al., 2014], a popular RL method for lifelong learning. TAO-PGELLA learns an actor-critic architecture using policy gradient. It is parameterized through fixed-size weights that are global for all tasks and task-specific weights (see Appendix A.6 for details).

We evaluate the performance of MOHITO and the baselines using the following performance measures.

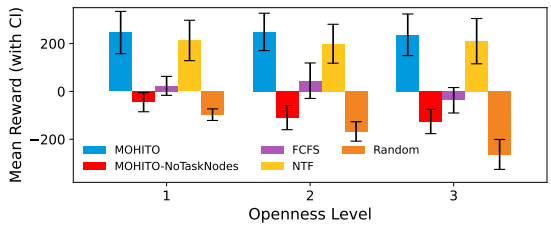
- *Episodic cumulative rewards*: the sum of rewards earned by all agents across testing episodes, measuring the effectiveness of agents at accomplishing their tasks.
- *Duration*: the timesteps taken to complete tasks, measuring the efficiency of agents at accomplishing their dynamically changing task set (ride-status shown for Rideshare).
- *Pooling* (Rideshare only): Time spent by agents multi-tasking by carrying multiple passengers simultaneously, a domain measure of efficiency with dynamic tasks.
- Fires Burned/Put-out (Wildfire Suppression only): The number of fires burned- or put-out. A direct domain-specific measure of task success or failure as a result of policy behavior.

6.4 EVALUATIONS

MOHITO improves on the baselines. Figure 5(a) presents the mean of cumulative rewards (with 95% confidence intervals) earned in Rideshare for the various openness levels. Similarly, mean rewards in Wildfire Suppression across all start states are shown in Fig. 5(b). MOHITO outperforms all baselines in each openness level with statistical significance (Wilcoxon signed rank tests, $p < .01$, Rideshare n=30, Wildfire Suppression n=60) for each domain.[1] We do not use TAO-PGELLA in Wildfire Suppression due to its poor performance in Rideshare and fundamental problems with adapting lifelong learning to task-open environments. Increased openness offers opportunity to service more passengers and risk when fighting more fires. Rewards generally increase with openness demonstrating that MOHITO learns to exploit additional task opportunities to increase its performance over the baselines.



(a) Rideshare - Episodic Rewards

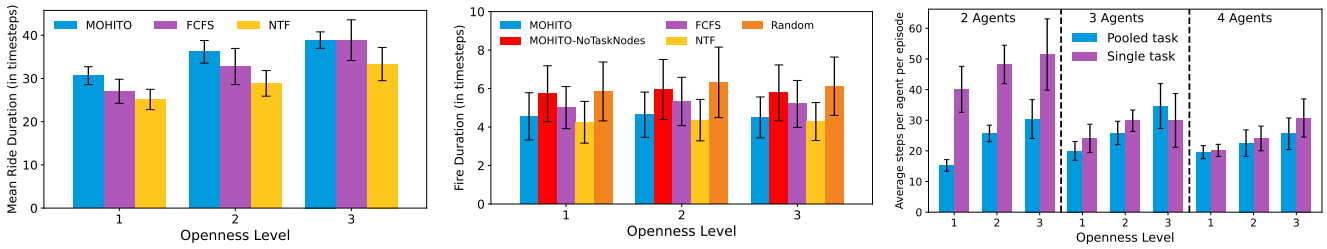


(b) Wildfire - Episodic Rewards

Figure 5: Mean episodic rewards (with 95% CI) (a) MOHITO performs well in Rideshare (b) Wildfire Suppression shows larger CI's likely from stochastic transitions.

MOHITO's policy is, in part, similar to NTF in Wildfire

[1]When all openness levels are considered together, MOHITO's performance remains significant except against NTF in Wildfire Suppression.



(a) Rideshare - Passenger Duration (b) Wildfire - Fire Duration (c) Rideshare - Pooling Efficiency

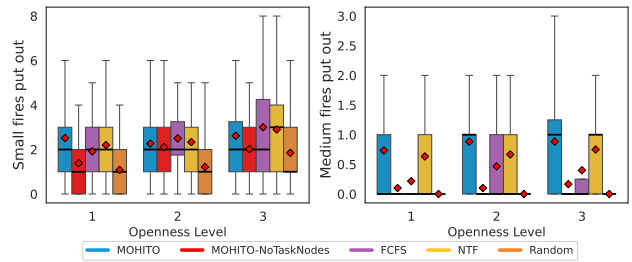
Figure 6: (a) Mean task duration shows passengers spending more time riding with MOHITO while they are pooled. (b) Mean task duration shows MOHITO only allowing fires to burn about as long as NTF. (c) The amount of time MOHITO spent carrying multiple passengers (pooling) vs carrying one or none.

Suppression but MOHITO earns a slightly higher reward than NTF because it puts out more medium fires compared to NTF (see Fig. 7a) in each openness level. As such, it displays better collaborative behavior between agents to tackle the most challenging tasks. Nonetheless, NTF is a strong baseline across our domains as it uses task-specific insight to select a greedy action.

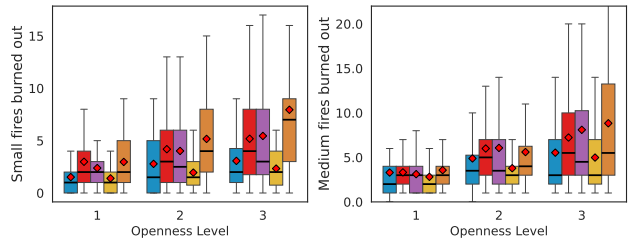
Task-action representation matters. In Fig. 5(b), we see MOHITO significantly outperform *MOHITO-NoTaskNodes*. The ablation's policy is not conditioned on task observations, so it cannot focus attacks on specific fires. Meanwhile, the shorter fire durations in Fig. 6(b) and fewer burnouts (see Fig. 7) indicate that MOHITO better manages risk by strategically distributing suppressant across fires to reduce burn-outs. This exemplifies the importance of learning a task preference in policies.

MOHITO can multitask. In Rideshare, we observe that MOHITO's converged policies lead to the complex phenomenon of *passenger pooling*. This involves picking up multiple passengers and conveying them efficiently to their destinations. We observe that the pooling behavior increases relative to single passenger rides at lower openness levels and higher agent counts in Fig. 6(c). The learned pooling behavior prioritizes the agent's driving efficiency by minimizing total drive distance, albeit at increased time cost as passenger service times are lengthened (cf., Fig. 6(a)). Such pooling behavior is the motivation for real-world services such as Uber Pool.

Lifelong learning underperforms. TAO-PGELLA exhibits poor ability to simultaneously manage existing tasks alongside new ones, resulting in poor rewards (Fig. 5(a)). Upon further investigation, we note that TAO-PGELLA often switches between tasks before completing a particular task. TAO-PGELLA learns policies that fit groups of tasks based on their features as a proxy for knowing all tasks a'priori – as required by the original PG-ELLA. However, the diversity of task characteristics due to task openness appears to lead to agent uncertainty about how to prioritize simultaneous tasks.



(a) Put Outs (more is better)



(b) Burn Outs (less is better)

Figure 7: Box plots of the numbers of fires (a) put out and (b) burned out by various methods per episode. The red dot indicates the mean. MOHITO has significantly fewer burn outs than *FCFS* and *MOHITO-NoTaskNodes*, even as the openness increases.

7 RELATED WORK

Lifelong/continual and multi-task learning also learn behavior that generalizes across changing tasks though they do not encompass fully the complexities of task openness.

Lifelong and continual learning do not choose tasks. Agents learn to improve their performance on future tasks by leveraging experiences with prior tasks [Chen and Liu, 2018]. Some methods use one shared parameterization of the agents' policy and try to avoid catastrophic forgetting as they learn from new tasks [Thrun and Mitchell, 1995]. Others separate knowledge learned from tasks then aggregate it. These are typically modeled as contextual MDPs

[Mendez et al., 2022, Kim et al., 2023]. One notable approach learns an implicit neural representation by having agents share task knowledge on a known schedule of tasks [Kolouri et al., 2023]. In others, agents interact with a sequence of single tasks drawn from a distribution [Abel et al., 2018]. Across lifelong learning, agents interact with the one present task and do not decide which task to prioritize.

There is a known distribution of tasks in multi-task learning. Agents learn from a set of tasks, typically fixed, with the goal of learning generalizable information across tasks [Tanaka and Yamamura, 2003, Varghese and Mahmoud, 2021]. Similar to lifelong learning, these methods often use a contextual MDP as the model [Sodhani et al., 2021, Andreas et al., 2017] with all contexts available to the agent. Unlike lifelong learning, multi-task learning can also be approached by more static models, a traditional MDP learned via ensemble learning [Rajeswaran et al., 2017], and a decentralized POMDP [Omidshafiei et al., 2017, Zhang et al., 2023]. Prior work also allows agents to explicitly prefer more rewarding tasks [Yu et al., 2023]. However, across all of these approaches, tasks are samples from a fixed known distribution of tasks.

Task openness is also addressed by methods for multi-agent path finding (MAPF), which often focuses on the use-case of robots servicing tasks in warehouses [Stern, 2022]. However, the majority of these methods adopt a fully centralized perspective to the joint problem and build on search techniques [Okumura, 2023]. Recently, RL has been explored as well for MAPF [Skrynnik et al., 2024], but agents do not explicitly reason about other agents' actions. RL is used only to navigate the agent to the assigned goal without collision. In contrast, agents using MOHITO reason about others' actions and choose tasks, which then makes pooling possible in Rideshare.

GNNs have been used for MARL previously. Jiang et al. [2020] and Rahman et al. [2021] contribute to RL in open multi-agent systems, both of which target agent openness without considering task openness. Both rely on the use of GNNs [Wang et al., 2016, Veličković et al., 2018] that adapt to changing input sizes [Hamilton et al., 2017]. These embed fully-connected coordination graphs and may intrinsically adapt to dynamic team sizes as agents depart or reenter. Our interaction hypergraphs significantly extend the coordination graphs to model tasks and actions, which offers a more expressive representation of the decision-making problem setting allowing its use for task openness.

8 CONCLUDING REMARKS

Learning how to act in environments with task openness, where the set of objectives or tasks is neither static nor predefined, is challenging. This paper presented three contributions towards agent reasoning under task openness: (1) a novel model, TaO-MG, using an interaction hypergraph to encode each agent's dynamic, open, and task-centric action space; (2) a task-open MARL approach, MOHITO, based on TaO-MG, which centrally learns to evaluate actions relative to all present tasks; and (3) two task-open domains, Rideshare and Wildfire Suppression, one competitive and the other cooperative, each with unique properties that serve as a testbed for exploring and evaluating approaches in this emerging topic. Our experiments demonstrated MOHITO's favorable ability to reason about task dynamics. This shows the sufficiency of the TaO-MG model and the viability and effectiveness of MOHITO in reasoning with task openness. Whereas such openness is a feature of many realistic problems, it may not be easily modeled using traditional games with fixed and bounded sets.

MOHITO has theoretical and practical limitations. The current TaO-MG model is limited to physical states that are perfectly observed. However, in domains such as Wildfire Suppression, partial observability is inherent and a key barrier to success. Practically, MOHITO may suffer from unstable training as tasks enter and leave, especially due to exogenous factors. The regularization component of the loss and utilizing graph norm helps toward this, but it may be challenging to identify convergence.

A key direction of future work is to allow considerations of agent openness as well alongside task openness. An approach would be to investigate ways of extending both TaO-MG and MOHITO to model and learn in the context of both these types of openness and the associated uncertainty.

Acknowledgements

This research was supported, in part, by a collaborative NSF Grant #IIS-2312657 (to Doshi), #IIS-2312658 (to Soh), and #IIS-2312659 (to Eck). Some of the computing occurred on the Holland Computing Center of the University of Nebraska, which receives support from the university's Office of Research and Economic Development and the Nebraska Research Initiative. We thank the graduate and undergraduate students who contributed to the baselines, the domains, and the testing of MOHITO: Ceferino Patino IV, Matthew Sentell, Alireza Saleh Abadi, and Tyler Billings.

REFERENCES

David Abel, Yuu Jinnai, Sophie Yue Guo, George Konidaris, and Michael Littman. Policy and value transfer in lifelong reinforcement learning. In *ICML*, pages 20–29. PMLR, 2018.

Haitham Bou Ammar, Eric Eaton, Paul Ruvolo, and Matthew Taylor. Online multi-task learning for policy gradient methods. In *ICML*, volume 32, pages 1206–1214. PMLR, 2014.

Jacob Andreas, Dan Klein, and Sergey Levine. Modular multitask reinforcement learning with policy sketches. In *ICML*, page 166–175, 2017.

Giacomo Bergami, Matteo Magnani, and Danilo Montesi. On joining graphs, 2016. URL `https://arxiv.org/abs/1608.05594`.

Wendelin Boehmer, Vitaly Kurin, and Shimon Whiteson. Deep coordination graphs. In Hal Daumé III and Aarti Singh, editors, *Machine Learning Research*, volume 119, pages 980–991. PMLR, 2020.

Samir Bouabdallah. *Design and control of quadrotors with application to autonomous flying*. PhD thesis, École Polytechnique Fédérale De Lausanne, 2007.

Craig Boutilier. Planning, learning and coordination in multiagent decision processes. In *TARK '96*, page 195–210, 1996.

D. Boychuk, W.J. Braun, R.J. Kulperger, Z.L. Krougly, and D.A. Stanford. A stochastic forest fire growth model. *Environmental and Ecological Statistics*, 16(2):133–151, 2009.

Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. *Reinforcement Learning and Dynamic Programming Using Function Approximators*. CRC Press, Inc., USA, 1st edition, 2010.

J. Calmet, A. Daemi, R. Endsuleit, and T. Mie. A liberal approach to openness in societies of agents. In *Engineering Societies in the Agents World IV, LNCS*, volume 3071, pages 81–92, 2004.

Muthukumaran Chandrasekaran, Adam Eck, Prashant Doshi, and Leenkiat Soh. Individual planning in open and typed agent systems. In *UAI'16*, pages 82–91. AUAI Press, 2016.

Zhiyuan Chen and Bing Liu. *Lifelong Reinforcement Learning*, pages 139–152. Springer International Publishing, 2018.

Jonathan Cohen, Jilles-Steeve Dibangoye, and Abdel-Illah Mouaddib. Open decentralized pomdps. In *IEEE 29th International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 977–984, 2017.

Adam Eck, Maulik Shah, Prashant Doshi, and Leenkiat Soh. Scalable decision-theoretic planning in open and typed multiagent systems. In *AAAI'20*, 2020.

Adam Eck, L.-K. Soh, and P. Doshi. Decision making in open agent systems. *AI Magazine*, 44(4):508–523, 2023.

Justin Gilmer, Samuel S. Schoenholz, Patrick F. Riley, Oriol Vinyals, and George E. Dahl. Neural message passing for quantum chemistry. In Doina Precup and Yee Whye Teh, editors, *Machine Learning Research*, volume 70, pages 1263–1272. PMLR, 2017.

Piotr J. Gmytrasiewicz and Prashant Doshi. A framework for sequential planning in multiagent settings. *JAIR*, 24: 49–79, 2005.

C. Guestrin, S. Venkataraman, and D. Koller. Context-specific multiagent coordination and planning with factored MDPs. In *Eighteenth National Conference on Artificial Intelligence*, page 253–259, 2002.

Tom Haider, Karsten Roscher, Felippe Schmoeller da Roza, and Stephan Günnemann. Out-of-distribution detection for reinforcement learning agents with probabilistic dynamics models. In *AAMAS '23*, page 851–859, Richland, SC, 2023. ISBN 9781450394321.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, page 1025–1035, 2017.

W. Jamroga, A. Meski, and M. Szreter. Modularity and openness in modeling multi-agent systems. In *GandALF*, pages 224–239, 2013.

Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph convolutional reinforcement learning. In *ICLR*, 2020.

J. Jumadinova, P. Dasgupta, , and L.-K. Soh. Strategic capability-learning for improved multi-agent collaboration in ad-hoc environments. *IEEE Trans. Systems, Man, & Cybernetics-Part A*, 44(8):1003–1014, 2014.

Anirudh Kakarlapudi, Gayathri Anil, Adam Eck, Prashant Doshi, and Leen-Kiat Soh. Decision-theoretic planning with communication in open multiagent systems. In *UAI*, pages 938–948, 2022.

S. Kim, L. F. Yang, and C.-A. Cheng. Provably efficient lifelong reinforcement learning with linear representation. In *ICLR*, 2023.

S. Kolouri, Ali Abbasi, Soroush Abbasi Koohpayegani, Parsa Nooralinejad, and Hamed Pirsiavash. Multia-agent lifelong implicit neural learning. *IEEE Signal Processing Letters*, 30:1812–1816, 2023.

Michael L. Littman. Markov games as a framework for multi-agent reinforcement learning. In *ICML*, pages 157–163. PMLR, 1994.

Lantao Liu and Gaurav S. Sukhatme. A solution to time-varying markov decision processes. *IEEE Robotics and Automation Letters*, 3(3):1631–1638, 2018. doi: 10.1109/LRA.2018.2801479.

Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *NIPS*, 2017.

J.A. Mendez, H. van Seijen, and E. Eaton. Modular lifelong reinforcement learning via neural composition. In *ICLR'2022*, 2022.

Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In Maria Florina Balcan and Kilian Q. Weinberger, editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mniha16.html.

Keisuke Okumura. Lacam: Search-based algorithm for quick multi-agent pathfinding. In *Proceedings of AAAI Conference on Artificial Intelligence (AAAI)*, 2023.

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

S. Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P. How, and John Vian. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. In *ICML*, 2017.

Melkior Ornik and Ufuk Topcu. Learning and planning for Time-Varying MDPs using maximum likelihood estimation. *J Mach Learn Res*, 22:1–40, February 2021.

Jan Peters and Stefan Schaal. Natural actor-critic. *Neurocomputing*, 71(7):1180–1190, 2008. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2007.11.026. URL https://www.sciencedirect.com/science/article/pii/S0925231208000532. Progress in Modeling, Theory, and Application of Computational Intelligenc.

Muhammad A Rahman, Niklas Hopner, Filippos Christianos, and Stefano V Albrecht. Towards open ad hoc teamwork using graph-based policy learning. In Marina Meila and Tong Zhang, editors, *Machine Learning Research*, volume 139, pages 8776–8786. PMLR, 2021.

Aravind Rajeswaran, Sarvjeet Ghotra, Balaraman Ravindran, and Sergey Levine. Epopt: Learning robust neural network policies using model ensembles. In *5th International Conference on Learning Representations ICLR*, 2017.

Paul Ruvolo and Eric Eaton. ELLA: An efficient lifelong learning algorithm. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of Machine Learning Research*, volume 28, pages 507–515, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR. URL https://proceedings.mlr.press/v28/ruvolo13.html.

Andreas Sedlmeier, Thomas Gabor, Thomy Phan, Lenz Belzner, and Claudia Linnhoff-Popien. Uncertainty-based out-of-distribution classification in deep reinforcement learning. In *ICAART'2020*, 2020. doi: 10.5220/0008949905220529. URL https://doi.org/10.5220%2F0008949905220529.

L. S. Shapley. Stochastic games*. *Proceedings of the National Academy of Sciences*, 39 (10):1095–1100, 1953. doi: 10.1073/pnas.39.10.1095. URL https://www.pnas.org/doi/abs/10.1073/pnas.39.10.1095.

O. Shehory. Software architecture attributes of multi-agent systems. In *1st International Workshop on Agent-Oriented Software Engineering, Revised papers*, pages 77–89. ACM, 2000.

A. Skrynnik, A. Andreychuk, M. Nesterova, K. Yakovlev, and A. Panov. Learn to follow: Decentralized lifelong multi-agent pathfinding via planning and learning. In *AAAI-24*, 2024.

Shagun Sodhani, Amy Zhang, and Joelle Pineau. Multi-task reinforcement learning with context-based representations. In *Proceedings of Machine Learning Research*, volume 139, pages 9767–9779. PMLR, 2021.

Roni Stern. *Multi-Agent Path Finding – An Overview*, page 96–115. Springer-Verlag, Berlin, Heidelberg, 2022. ISBN 978-3-030-33273-0. URL https://doi.org/10.1007/978-3-030-33274-7_6.

F. Tanaka and M. Yamamura. Multitask reinforcement learning on the distribution of mdps. In *IEEE International Symposium on Computational Intelligence in Robotics and Automation*, volume 3, pages 1108–1113 vol.3, 2003.

J Terry, Benjamin Black, Nathaniel Grammel, Mario Jayakumar, Ananth Hari, Ryan Sullivan, Luis S Santos, Clemens Dieffendahl, Caroline Horsch, Rodrigo Perez-Vicente, et al. Pettingzoo: Gym for multi-agent reinforcement learning. *NeurIPS*, 34:15032–15043, 2021.

S. Thrun and T.M. Mitchell. Lifelong robot learning. *Robotics and Autonomous Systems*, 15(1-2):25–46, 1995.

Uber. Airport Queue Access, 2025. URL https://help.uber.com/en/driving-and-delivering/article/queue-access?nodeId=1275528f-2ba7-4ca4-982f-6aca4e3fad99. Accessed: 2025-01-11.

N.K. Ure, S. Omidshafiei, B.T. Lopez, A.-A. Agha-Mohammadi, J.P. How, and J. Vian. Online heterogeneous multiagent learning under limited communication with applications to forest fire management. In *IROS*, pages 5181–5188, 2015.

U.S. Fire Admin. *Risk Management Practices*. Federal Emergency Management Agency, 2018. URL https://www.usfa.fema.gov/downloads/pdf/publications/risk_management_practices.pdf. Accessed: 1/15/25.

Nelson Vithayathil Varghese and Qusay H. Mahmoud. A hybrid multi-task learning approach for optimizing deep reinforcement learning agents. *IEEE Access*, 9:44681–44703, 2021. doi: 10.1109/ACCESS.2021.3065710.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018. URL https://arxiv.org/abs/1710.10903.

Daixin Wang, Peng Cui, and Wenwu Zhu. Structural deep network embedding. In *KDD '16*, page 1225–1234, 2016.

R. J. Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine Learning*, 8:229–256, 1992.

Y. Yu, Q. Yin, J. Zhang, and K. Huang. Prioritized tasks mining for multi-task cooperative multi-agent reinforcement learning. In *AAMAS*, pages 1615–1623, 2023.

Fuxiang Zhang, Chengxing Jia, Yi-Chen Li, Lei Yuan, Yang Yu, and Zhongzhang Zhang. Discovering generalizable multi-agent coordination skills from multi-task offline data. In *ICLR*, 2023.

MOHITO: Multi-Agent Reinforcement Learning using Hypergraphs for Task-Open Systems (Supplementary Material)

Gayathri Anil[1] **Prashant Doshi**[1] **Daniel Redder**[1] **Adam Eck**[2] **Leen-Kiat Soh**[3]

[1]THINC Lab, School of Computing, University of Georgia, Athens, GA, USA
[2]Computer Science Department, Oberlin College, Oberlin, Ohio, USA
[3]School of Computing, University of Nebraska, Lincoln, Nebraska, USA

A TECHNICAL APPENDIX

A.1 SPACE COMPLEXITY

Here we evaluate the space complexity of MOHITO.

A.1.1 Interaction and observation graphs

Consider an arbitrary graph g with nodes with feature size f and edge feature size f^e. The space used to store g is the sum of the space used to store g' s edges, g^e, and g' s nodes, g^n, Eq. 4.

$$\begin{aligned} |g| =& (|g^n| \cdot f) \\ &+ (2 \cdot |g^e| + |g^e| \cdot f^e) \end{aligned} \tag{4}$$

Now we use Eq. 4 to find $\mathcal{O}(|G_{\mathbf{C}}|)$, the size of MOHITO's critic input. Recall that $\mathbf{G_O} \subseteq G_{\mathbf{C}}$, so $|\mathbf{G_O}| = \mathcal{O}(|G_{\mathbf{C}}|)$ thus we focus on the critic graph rather than, $\mathbf{G_O}$, the observation graphs. Both our interaction graphs are composed of **task observations** $\mathcal{X}$, **action descriptions** $\mathcal{A}$, **agent observations** $\mathcal{N}$, and **hyperedge nodes** E (i.e. $|G^n_{\mathbf{C}}|$ as seen in Eq. 5). In Eq. 5, f is the same for all $|G^n_{\mathbf{c}}|$ as all nodes are padded to the largest.

$$|G^n_{\mathbf{C}}| = |\mathcal{A}| + |\mathcal{N}| + |\mathcal{X}| + |E| \tag{5}$$

The environment provides us the number of agents and tasks, so $|\mathcal{N}| = |Ag|$ and $|\mathcal{X}| = |X|$. Let A_i be the largest agent action space. The number of action nodes is bounded by the number of unique actions, so $|\mathcal{A}| = \mathcal{O}(|Ag| \cdot |A_i|)$. Each task-action is a relationship between a $X \in \mathcal{X}$ and a $a \in \mathcal{A}$, and task-action nodes, E, represent an agents relationship to that task-action. Thus $|E|$ is bounded by $\mathcal{O}(\cdot|\mathcal{A}| \cdot |\mathcal{N}| \cdot |\mathcal{X}|)$.

Substituting $|\mathcal{A}|, |\mathcal{N}|, |\mathcal{X}|, |E|$, and Eq. 5 into Eq. 4,

$$\begin{aligned} &|G_{\mathbf{C}}| \leq f(|\mathcal{A}| + |\mathcal{N}| + |\mathcal{X}| + |E|) + 6(|\mathcal{A}| \cdot |\mathcal{N}| \cdot |\mathcal{X}|) \\ &|G_{\mathbf{C}}| = \mathcal{O}(f \cdot |Ag|^2 \cdot |A_i| \cdot |X|) \end{aligned} \tag{6}$$

A.1.2 MOHITO execution

MOHITO actors are comprised of #layer-many graph attention transformers (GAT) Veličković et al. [2018] layers, activated by ReLU layers, and the hyperedge, ed, is selected by ArgMax. GAT scale linearly in $|G^n_{\mathbf{C}}|$ and $|G^e_{\mathbf{C}}|$ Veličković et al.

[2018]. Here we assume they scale polynomially in input and output feature size, f, f', following their defined single head GAT time complexity Eq. 7.

$$GAT^{time} = \mathcal{O}(|G^n_{\mathbf{C}}| \times f \times f' + |G^e_{\mathbf{C}}| * f') \tag{7}$$

Within MOHITO, $f' = \{f, \text{hidden dim}, \text{hidden dim } \times \text{\#heads}\}$ across layers. We substitute $|G^e_{\mathbf{C}}|$, $|G^n_{\mathbf{C}}|$, and simplify with Eq. 7 to get MOHITO (M) execution space complexity Eq. 8.

$$\begin{aligned} M^{space}_{exe} &\leq \ \text{\#heads}(f^2 \cdot |Ag|^2 \cdot |A_i| \cdot |X| + f|E|) \\ M^{space}_{exe} &= \mathcal{O}(\text{\#heads} \times f \times |G_{\mathbf{C}}|) \end{aligned} \tag{8}$$

A.1.3 Critic loss computation, L_Q

The critic performs global mean pooling and linear layers subsequent to the prior actor structure, which are $\mathcal{O}(f \times |G^n_{\mathbf{C}}|)$, and $\mathcal{O}(f \times |Ag|)$, respectively. Both are strictly dominated by $|M^{space}_{exe}|$ because $f \in \mathbb{N}$. The only additional concurrent space requirement is other agents' choosen hyperedges, $ed \in E$, $[ed, ...ed_n]$, term two in Eq. 9.

$$\begin{aligned} critic_{space} &\leq (\text{\#heads} \times f \times |G_{\mathbf{C}}|) + (f \times |Ag|) \\ critic_{space} &= \mathcal{O}(\text{\#heads} \times f \times |G_{\mathbf{C}}|) \end{aligned} \tag{9}$$

In L_Q, we perform two passes $Q^{\boldsymbol{\phi(j)}}_i, Q^{\boldsymbol{\phi}'}_i(j)$ per agent per experience in the batch. The size of the Q-values in the output is 1, and $|r^j_i| = 1$. The space complexity of storing this loss is dominated by $critic_{space}$ and the batched inputs. We discuss the upper bound of batch size in Appendix A.7. Removing dominated terms we get the space complexity of critic loss, Eq. 9.

$$M^{space}_{L_Q} = \mathcal{O}\Big(M^{space}_{exe} + \text{batch size} \times \big(|G_{\mathbf{C}}| + (|Ag| \times f)\big)\Big) \tag{10}$$

A.1.4 Actor loss computation, L_π

As with L_Q, the regularization only takes a multiple of parameter space, $\boldsymbol{\theta}$ additional space here. Here we calculate $Q^{'\boldsymbol{\phi}}_i(j)$ using the updated main network $\boldsymbol{\phi}$. We do not need the other agents' chosen hyperedges, we need $\mathbf{G_O}$ per batch, and with the assertion that $\mathbf{G_O} \subseteq G_{\mathbf{C}}$, we use the same bound as the critic, Eq. 11.

$$M^{space}_{L_\pi} = \mathcal{O}(M^{space}_{L_Q}) \tag{11}$$

A.2 TIME COMPLEXITY

Here we evaluate the time complexity of hypergraph construction, policy query, and loss.

A.2.1 Interaction and observation graphs

We have pettingzoo Terry et al. [2021] environments, so it is important to consider the additional time to convert a gymnasium space observation to an interaction graph. Alg. 2 shows our interaction graph construction. identifying unique actions is $\mathcal{O}\left((|Ag| \times |X| \times |A_i|)log(|Ag| \times |X| \times |A_i|)\right)$, and concatenation is $\mathcal{O}(|Ag|^2 + |X| + |A_i|)$. Here the common case is linear w.r.t. $|Ag|$ where agents frequently have the same rather than distinct actions available to them, but the worst case is where all agents have unique actions thus,

$$G^{time}_{\mathbf{C}} = \mathcal{O}(|Ag|^2 \times |X| \times |A_i|) \tag{12}$$

Algorithm 2 Interaction Graph Construction

Input: $agents < |Ag|, f >$,
$tasks < |X|, f >$,
$actspace[Ag, X] \rightarrow [A]$
Output: G_O or G_C ▷ G_C is returned if all tasks and the union of the action space is given.

```
actions ← UNIQUE(actspace.values)
nodes ← Concatenate(agents, tasks, actions)
edges ← []
taskActions ← 0
for agent ∈ agents.index do
    for task ∈ tasks.index + |agents| do
        thisActIndex ← actspace[agent, task].index
        for action ∈ thisActIndex + |agents| + |tasks| do
            hyperedge ← (agent, task, action)
            hyperedge ← pad(hyperedge, f)
            APPEND(nodes, hyperedge)
            APPEND(edges, [agent, hyperedge], [task, hyperedge], [action, hyperedge])
return (nodes, edges)
```

A.2.2 MOHITO execution

In Appendix A.1, we assume that the time complexity of a single head GAT, Eq. 7, is equivalent to its space complexity. We make the same assumptions on the value of f, f' here to get a time upper bound for any layer in the GAT; $f, f' \leq f$. To query π we multiply by the number of layers because, with our upper bound, time scales linearly with more layers, Eq. 13.

$$M_{exe}^{time} = \mathcal{O}(\text{\#layers} \times \text{\#heads} \times f \times |G_{\mathbf{C}}|) \tag{13}$$

A.2.3 Critic loss

The time complexities of the mean pooling and linear layer are the same as the space complexity mentioned in, Appendix A.1; $\mathcal{O}(f \times |G_{\mathbf{C}}^n|)$, and $\mathcal{O}(f \times |Ag|)$ respectively. The only difference here is accounting for all Q passes instead of space growth, Eq. 14

$$\begin{aligned} critic_{time} =& \mathcal{O}(M_{exe}^{time}) \\ M_{L_Q}^{time} =& O\big(\text{batch size} \times (M_{exe}^{time} + f \times (|G_{\mathbf{C}}^n| + |Ag|)\big) \end{aligned} \tag{14}$$

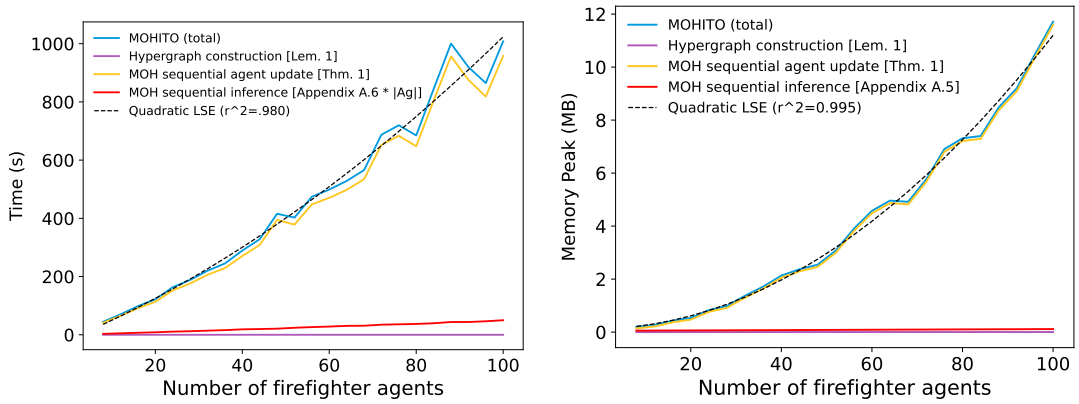
A.2.4 Actor loss

As in Appendix A.1, these losses must be calculated over seperate runs, $L_Q(..., \phi)$ and $L_\pi(..., \phi')$. Then mean square error is calculated which is trivially $\mathcal{O}(\text{batch size})$.

$$\begin{aligned} M_{L_\pi}^{time} &\leq (M_{exe}^{time} * 2 + \text{batch size}) \\ M_{L_\pi}^{time} &= \mathcal{O}(M_{exe}^{time} + \text{ batch size}) \end{aligned} \tag{15}$$

A.3 MOHITO PROFILING

Results confirm MOHITO's complexity from Thm. 1. We scale the second starting state to include linearly more agents and tasks. We execute MOHITO from 8 to 100 agents in increments of 4 agents and 3 tasks. This is run on a Linux virtual machine with 4 AMD EPYC 7302 16-Core CPUs and 500GB of DDR4 RAM. We show peak memory usage, Fig. 8(a), and time, Fig. 8(b), to run MOHITO. These fit to a quadratic curve and empirically show our success in avoiding exponential complexity.



(a) Time profiling on Wildfire Suppression. The largest time usage is i/o on unpacking batches during training rather than inference time.

(b) Peak memory usage from profiling Wildfire Suppression. The sequential agent update tracks closely to the total usage.

Figure 8: Results from profiling MOHITO align with Appendix A.2.2.

A.4 MOHITO PARAMETERS AND STRUCTURE

Here we show all parameters used for MOHITO in our two domains. In Table 1, we show all parameters we used in the experiments shown in this paper.

Table 1: Parameters For MOHITO training

MOHITO Parameters	**rideshare**	**wildfire**
γ	0.9	0.99
#GAT Layers	20	3
hidden dim	50	24
grad_clip: max norm	5	0.0
Environment		
training seed	16	16
max steps per episode	100	100
$\|Episodes\|$	20,000	20,000
K: batch size	20	16
B: buffer size	0	1,000
Graph Norm	yes	no
validation frequency	every 5 model updates	10 model updates
Actor		
learning rate	0.001	0.009
λ_A: regularization coefficient	0.1	0.01
ψ_A: coefficient for slow-update	0.05	0.005
Critic		
learning rate	0.01	0.01
λ_Q	0.01	0.0
ψ_Q	0.05	0.005
Actor Hyperedges	MLP, Fig. 10(a)	Pooling, Fig. 10(b)

The architecture of the actor, Fig. 9 and critic, Fig. 10, are similar up to their final layers. They both contain 2-head GATConv, activation (ReLU), and dropout layers per **#GAT Layers**. The critic considers agent actions through incorporating their hyperedge features. Either through a MLP at the end of the network (as seen in Fig. 10(a) and Rideshare), or incorporating all hyperedges into the critic graph and pooling over the selected (as see in Fig. 10(b) and Wildfire Suppression).

In Fig. 11, we show MOHITO validation rewards in Rideshare across 3000 episodes. The set of tasks given to MOHITO in training is fixed and different from those used in validation. We trained for a fixed duration for each openness level and agent number then picked the best performing policies for Rideshare.

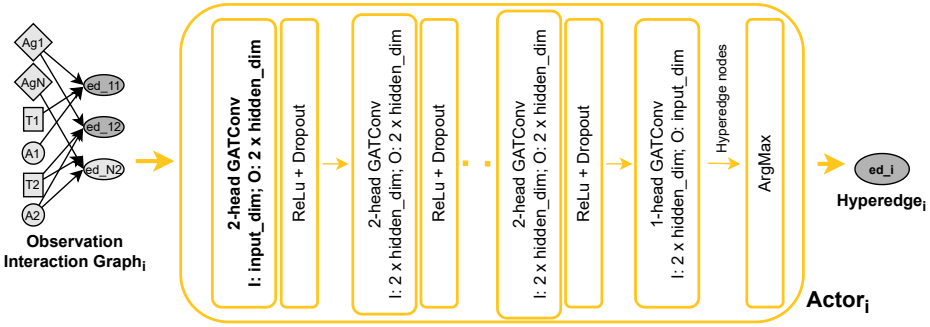
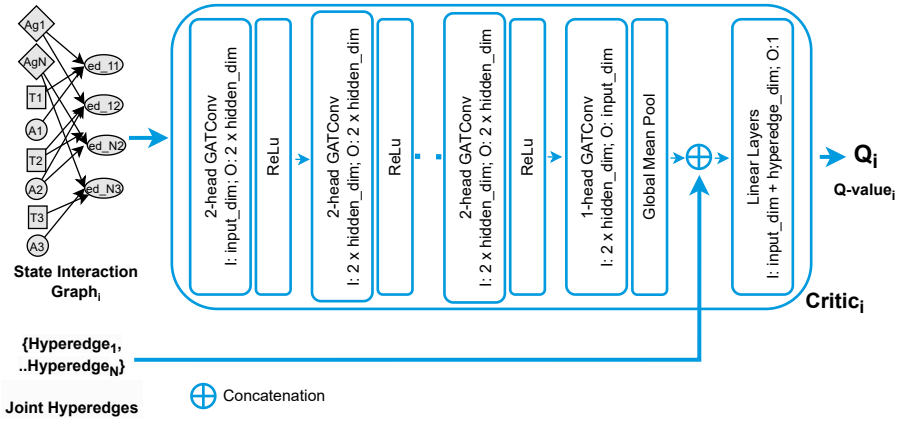
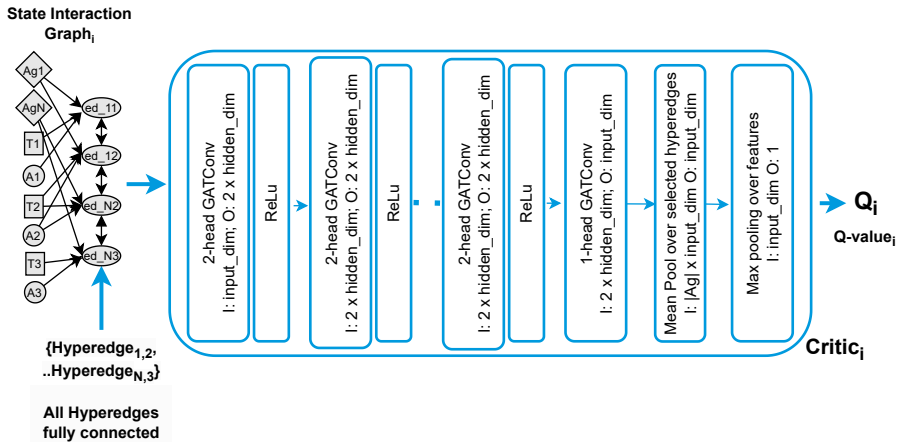


Figure 9: Actor network architecture. It is a series of graph attention transformer layers followed by ReLU activation and dropout. When we use graph norm, it is done before ReLU.



(a) The MLP based critic architecture for MOHITO. This addresses the unbounded number of task-actions by including agent input only from selected actions.



(b) The pooling based architecture for MOHITO. This incorporates all actor hyperedges into the state interaction graph. This addresses the unbounded number of task-actions by keeping the Q-value calculation within the graph attention transformers.

Figure 10: The critic architecture for MOHITO. It is a series of graph attention transformers activated by ReLU and normalized by graph norm at all but the last layer.

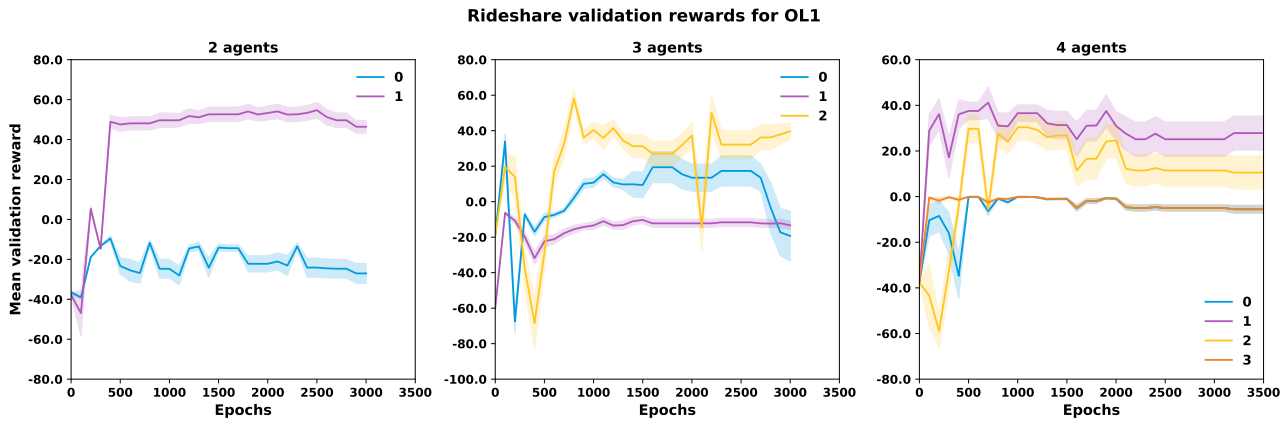


Figure 11: Some agents earn 0 rewards. This is a common optima for the system we encountered where a subset of agents choose to do nothing, and other agents dominate the environment.

A.5 TAO-MG BASED DEFINITIONS OF DOMAINS

A.5.1 Wildfire Suppression

Here we formally define Wildfire Suppression, as a TaO-MG, and we define a general approach to represent task-open action spaces.

$$\text{TaO-MG} \stackrel{def}{=} \langle M, X, \Psi \rangle$$

X is the current set of lit fires. The features of each fire are,

$$x = [position, int_{initial}, fType_x]$$

M is a Markov game representation of for a given set of fires X.

- **Ag** is a fixed set of agents. In this work we show a domain with two firefighters. These agents leave the environment when they run out of suppressant, and as they No-Op, they stochastically regain suppressant and eventually return to the environment. Here leaving/returning simply effects whether they can act on fires. No new agents appear in this domain's experiments.
- **S** is a set of 4 matrices with the shape of the grid, $\{fType, int, ag, sup\}$. $fType$ contains integers where non-zero elements are burnable tiles. $fType$ corresponds to the number of agents which must collaborate to stochastically fight and put out the fire. int represents the intensity of each fire $[0, n]$ at time t. ag represents the location of each agent. sup represents the amount of fire suppressant each agent has at time t.
- **A** is the Cartesian product of all agent action spaces A_i. An agent can fight any available fire if they can reach it (via the Chebyshev distance) and they have enough suppressant. Meanwhile, an agent can always No-Op.

  In this work we handle task-open action spaces using hypergraphs, but to provide a more general view we next define agent action spaces using two simpler approaches. Consider an arbitrary, discrete action, task-open, environment env. In this environment, there exists tasks X such that each task is associated with $[0, \infty)$ actions per agent, $A^t_{x,i}$. The full action space for agent i at time t must identify which action is being taken and on which task it is being taken. We can enumerate the task-action by shifting the value of each action by the size of the prior task action spaces Eq. 16, or we explicitly represent action selection as a two-part process: select the task first and then the action, i.e., Eq. 17.

$$A^t_i = \bigcup_{x \in X} \begin{cases} A^t_{x,i} & x = 0 \\ A^t_{x,i} + \sum_{x'=X_0}^{x-1} |A^t_{x',i}| & x > 0. \end{cases} \tag{16}$$

$$A_i^t = \bigcup_{x \in X} (x, A_{t,x,i}). \tag{17}$$

In our examples each task has a single action at any time step, so Eq. 16 is equivalent to Eq. 17 in space requirement. However there are benefits to both, Eq. 17 is more interpretable for an arbitrary domain with more actions, and Eq. 16 is convenient for environment implementation. Wildfire Suppression uses the explicit action representation and Rideshare uses the implicit action representation.

- $\mathbf{T}$ transitions in wildfire are stochastic and occur on int, and sup. Fire spread is handled by a 2d convolutional filter over where $int_{N,S,E,W}$ is determined by the wildfire spread model and scales with the base spread parameter [Boychuk et al., 2009, Ure et al., 2015]. Note that this convolution applies over the existence of lit fires, so the number of fires in a cell doesn't impact fire spread. Additionally there is a 0.1 random ignition probability present in all cells.

$$conv = \begin{bmatrix} 0.0 & int_N^+ & 0.0 \\ int_W^+ & 0.1 & int_E^+ \\ 0.0 & int_S^+ & 0.0 \end{bmatrix}, \text{2d conv w/o bias} \tag{18}$$

Suppressant transitions are determined by whether the agent in question fought a fire, $sup^- = 1/3$, then it decreases by 1, or No-Op, $sup^+ = 1/2$ where it will be set to its maximum value, 2.

Intensity transitions are determined separately from fire spread. If enough agents have fought a fire $|attacks| \geq fType$ the fire intensity decreases stochastically by 1, Eq. 19.

$$p(int_x^-) = 0.8 + .12\,(|attacks_x| - ftype_x) \tag{19}$$

Otherwise, fire intensity will increase deterministically, $int^+ = 1.0$, unless the fire is about to burn out ex $n-1 \to n$. In such a case, fire intensity will increase stochastically, $p(int^+, int_{burn}^+)$ where $int_{burn}^+ = .2238$ is calculated from the fire spread model (not parameterized by base_spread).

- $\mathbf{R}$ rewards are joint in wildfire. While there exists penalties for fighting a not-present fire or fighting a fire without suppressant, these do not occur because their tasks are not present in such cases. A reward of 400 is given for putting out the collaborative center fire; 20 is given for putting out any other fire, and a penalty of -10, -25 for a small or medium fire burning out.

$\mathbf{\Psi}$ is the generator function which converts $M \to M'$ when X changes. S, T, A, and R change when $X \to X'$.

- $\mathbf{S}$ changes when a new task, x, enters the environment its initial intensity and fire type are determined by its position. *accelerated* cells produce dangerous medium fires that are at state 3, other cells produce small intensity 2 fires. When a task leaves the environment that fire will not be included in the action space of any agent. While a theoretically infinite number of fires can enter the environment in each cell, we cap the number at 13 for practicality.
- $\mathbf{A}$ is updated as the union of present task action spaces.
- $\mathbf{T}$ is updated to exclude burned out fires, and include newly lit fires. $fire_spread$ changes according to which fires are now present in the environment, with additional lit cells raising the probability of adjacent fire spread.
- $\mathbf{R}$ a new task, x, introduces the possibility of new rewards from its future exit whether that be extinguished or burned-out fires. When a task x leaves, its reward is removed until it is reignited.

A.5.2 Rideshare

Here we formally define Rideshare as a TaO-MG. First we define X, the underlying Markov game representation, M, and then we address how we update M to account for a new M, Ψ.

$$\text{TaO-MG} \stackrel{def}{=} \langle M, X, \Psi \rangle$$

$\mathbf{X}$ is the current set of passengers. Passenger features, see Eq. 20, are: $pick_loc$ is the initial pick-up location where a task x spawned. $drop_loc$ is the destination of the passenger. $ride_fare$ is the rewards earned for dropping this passenger off, determined by Eq. 21 using the Manhattan distance.

$$x_{i,j} = [pick_loc, drop_loc, ride_fare] \tag{20}$$

$$\begin{aligned} ride_fare = 3 \times max[3, dist(pick_loc, drop_loc)] \\ + rand[-1, 2] \end{aligned} \tag{21}$$

Passengers can also be in one of three states, referred to as $ride_status$ in the paper:

- Ride request: A new unassigned task that can be accepted.
- Assigned passenger: A passenger who has been assigned and can be picked up by the assigning agent.
- Rider: A passenger riding with an agent who can be dropped off.

This task state is stored by its index: P_i is the ith ride request, $P_{j,i}$ is the ith accepted passenger for the jth agent if it is not stored with a agent in s. We define the rate that $x \rightarrow X'$ in rideshare setup 6.1.

$\mathbf{M}$ is a Markov game representation for a given set of passengers X.

- $\mathbf{Ag}$ is a fixed set of agents. In this work we considered 2-, 3-, and 4-agent configurations.
- $\mathbf{s}$ is represented using a matrix of vectors (see below), where the grid cells indicate the agent and task locations. All values of these vectors are static task features.

$$s = \begin{bmatrix} (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{Ag_3\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{\},\{P_1\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{\},\{\}) & (\{Ag_2\},\{P_{21}\}) \\ (\{Ag_1\},\{P_{11},P_{12}\}) & (\{\},\{\}) & (\{\},\{\}) \\ (\{\},\{\}) & (\{\},\{\}) & (\{\},\{\}) \end{bmatrix}$$

- $\mathbf{A}$ is the Cartesian product of all agent action spaces A_i where:

$$A_i = \text{No-Op} \bigcup_{x \in X} \begin{cases} \text{accept}_x & \text{x is ride_request} \\ \text{pick_up}_x & \text{x is accepted by i} \\ \text{drop_off}_x & \text{x is i's passenger} \end{cases}$$

- $\mathbf{T}$ is the transition function which determines agent and passenger movement. All movement transitions are deterministic. Agents who use pick_up$_x$ or drop_off$_x$ move one cell (with diagonals) closer to x, direction determined by A*. When an agent lands on the same cell as x, with a pick_up$_x$, x will become a *passenger*. If an agent lands on $drop_loc_x$ with drop_off$_x$ then x is removed from X. Agents who use $accept_x$ do not move, and x is added to the set of riding passengers for that agent.
- $\mathbf{R}$ is the agent-wise function determining rewards based on the joint action a, the ith agent, and tasks X. We define this in pieces for simplicty.

$$\begin{aligned} \text{fare}_i &= \begin{cases} ride_fare_x & a_i = \text{ drop_off}_x \wedge x \notin X' \\ 0 & a_i \neq \text{drop_off}_x \vee x \in X' \end{cases} \\ \text{move_cost}_i &= \begin{cases} -1.2 & agent_pos \neq agent_pos' \\ 0 & agent_pos = agent_pos' \end{cases} \\ \text{pick_cost}_i &= \begin{cases} -0.1 & a_i = \text{ pick_up}_x \wedge agent_pos = x_pos \\ 0.0 & a_i \neq \text{pick_up}_x \vee agent_pos \neq x_pos \end{cases} \\ \text{pool_limit_cost}_i &= \begin{cases} -2 & a_i = \text{pick_up}_x \wedge |x \in i| > 2 \\ 0 & a_i \neq \text{pick_up}_x \vee |x \in i| \leq 2 \end{cases} \end{aligned} \tag{22}$$

An additional wait_cost, -2, is added to all agents if any x hasn't changed $ride_status$ for a count of steps (accept:5, pick_up:10, drop_off:10).

The final cost, unserved_cost = $-0.5\times$ (open seats), is applied if the number of unaccepted passengers is greater than the total number of seats across all passengers. This, together with the wait cost, discourage model inaction.

$$\begin{aligned} r_i = & fare_i + move_cost_i + pick_cost_i \\ & + pool_limit_cost_i + wait_cost + unserved_cost \end{aligned} \tag{23}$$

$\mathbf{\Psi}$ is the generator function which converts $M \rightarrow M'$ when X changes. S, T, and R change when $X \rightarrow X'$.

- $\mathbf{S}$ when new passengers, x, enter, x is added to the second set of the x_pos element of s to get s'. Then S' is the Cartesian product of s' across possible pos for all $x \in X$ and $i \in Ag$.
- $\mathbf{T}$ maintains the same behavior as described prior, but when explicitly defined, it is updated to only include transitions for S' removing the transitions for departing passengers and adding new.
- $\mathbf{R}$ the reward function is redefined to account for new waiting tasks for $wait_cost$ and $unserved_cost$. Additionally, any departing passengers are removed from the reward function.

A.6 TAO-PGELLA

First described by Ammar et al. [2014], PG-ELLA is an extension of the authors' previous work in linear regression [Ruvolo and Eaton, 2013]. It is a lifelong learning algorithm and features a modular structure capable of utilizing learning a variety of base learners [Williams, 1992, Peters and Schaal, 2008] to individually learn policies for like tasks such as cartpole and quadrotor [Busoniu et al., 2010],[Bouabdallah, 2007]. Information is shared between tasks through a large latent space L which combines with task specific parameters s_x to form task-specific policy weights $\theta_x = Ls_x$.

PG-ELLA falls short of open-task applications in three key aspects. Multi-Agency: PG-ELLA is a single agent model; (2) Dynamic Spaces: The action and observation space of the environment change as X changes. L has a fixed size and cannot handle changing shape inputs; (3) Simultaneous tasks: The generation of trajectories depends on a single task present in the environment at a time. We next describe how we address these challenges, and we show them in Fig. 12.

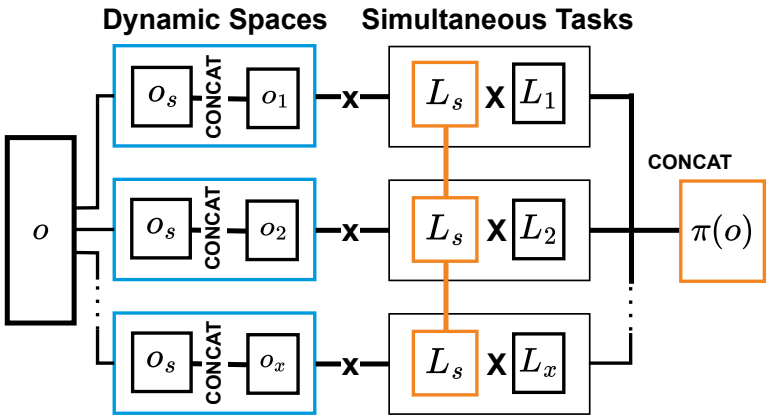


Figure 12: Network diagram of TaO-MG, shows how we handle dynamic spaces with shared/individual observations, and simultaneously present tasks with task-specific policy concatentation.

TAO-PGELLA's base learners, such as reinforce [Williams, 1992] have existing effective extensions to multi-agent environments. We use advantage actor critic (A2C) [Mnih et al., 2016]. Rideshare observations can be factored into task-generic, O_s, and task-specific observations, $[o_1, ..., o_x]$;

- *task-generic*: my position, the position of other drivers, the number of passengers I have accepted, and the number of passengers riding with me.
- *task-specific*: the position of a passenger, the destination of a passenger, the fare for that passenger, how long a passenger has waited.

We concatenate (O_s, o_x) whose shape is now the fixed input size of $\theta_x \forall x$ to solve the dynamic shape challenge. Simultaneous tasks require changes in both representation and training. We consider unique tasks as individual tasks rather than X as one task (the space of X is unbounded in task openness), and in the case of an unbounded number of possible individual tasks, such as continuous fares in Rideshare, we discretize the space, grouping similar tasks (i.e., $x \in \chi \subset X$).

We select an action from TAO-PGELLA's actor by querying all task-specific parameters with present tasks, $\theta_x \iff x \in X$, then perform a softmax over their concatenated logits to select a task-action.

Algorithm 3 TaO-PGELLA (k, λ, μ)

1: $|\chi| \leftarrow 0$
2: $A \leftarrow \mathbf{zeros}_{k \times d, k \times d}$
3: $b \leftarrow \mathbf{zeros}_{k \times d, 1}$
4: $L \leftarrow \mathbf{zeros}_{d,k}$
5: **while** some task is available **do**
6: $\quad L \leftarrow \text{reinitializeZeroColumns}(L)$
7: $\quad (\mathbb{X}, \mathbb{R}) \leftarrow \text{getTrajectories}(\theta)$ ▷ Online interaction and ϵ-greedy
8: $\quad \langle o_1, ..., o_x, O_s \rangle \leftarrow \text{getTasks}(\mathbb{T}, \mathbb{R})$
9: $\quad$ **for** $o_j \in \langle o_1, ..., o_x \rangle$ **do**
10: $\quad\quad (\mathbb{X}_{(x_j)}, \mathbb{R}_{(x_j)}) \leftarrow \text{filterTrajectories}(\mathbb{X}, \mathbb{R}, x_j)$
11: $\quad\quad$ **if** $\text{isNewTask}(x_j)$ **then**
12: $\quad\quad\quad |\chi| \leftarrow |\chi| + 1$
13: $\quad\quad$ **else**
14: $\quad\quad\quad A \leftarrow A - (s_{(x_i)} s_{(x_i)^\top}) \otimes \Gamma_{(x_i)}$
15: $\quad\quad\quad b \leftarrow b - \text{vec}(s_{(x_i)^\top} \otimes (\theta_{x_j}^\top \Gamma_{(x_j)}))$
16: $\quad\quad$ compute $\theta_{(x_j)}$ and $\Gamma_{(x_j)}$ from $(\mathbb{X}_{(x_j)}, \mathbb{R}_{(x_j)})$
17: $\quad\quad s_{(x_j)} \leftarrow \text{argmin}_s \ell(L, s, \theta_{x_j}, \Gamma_{(x_j)})$ ▷ optimize local learner by adam SGD
18: $\quad\quad A \leftarrow A + (s_{(x_j)} s_{(x_j)}^\top) \otimes \Gamma_{(x_j)}$
19: $\quad\quad b \leftarrow b + \text{vec}(s_{(x_j)}^\top \otimes (\theta_{(x_j)}^\top \Gamma_{(x_j)}))$
$\quad$)
20: $\quad L \leftarrow \text{mat}((\frac{1}{X} A + \lambda I_{k \times d, k \times d})^{-1} \frac{1}{|\chi|} b)$

Algorithm 3 presents the algorithm for TAO-PGELLA. We start by initializing how many x we have encountered, $|\chi|$, and the matrices used with the Hessian to calculate the updated L later (lines 1-4). *reinitializeZeroColumns* resets all zero columns, $L_c \sim \text{Uniform}[-1, 1]$. We interact with the environment given our current θ to obtain trajectories, $(\mathbb{X}, \mathbb{R})$, exploring with ϵ-greedy (line 7). Next, we perform the parameter updates from PG-ELLA Ruvolo and Eaton [2013] looping over all unique $x \in \mathbb{X}$ (lines 9-20). *filterTrajectories* constructs $\mathbb{X}_j, \mathbb{R}_j \leftarrow \{x, r | x \in \mathbb{X}\}$ where $x \in \mathbb{X}$ means that the action taken in that trajectory is associated with task x and task x is observed.

A.7 TABULAR RL COMPLEXITY

In this work we proposed a deep RL solution to TaO-MG environments. Here we consider the complexity of MOHITO if we use classic RL lookup tables in place of our π and Q GNNs. We also consider the upper bound for MOHITO batch size.

A.7.1 MOHITO lookup space

To encode actions conditioned on $\mathbf{G_O}$, we make the following assumptions:

1. Ψ_A is deterministic such that $E = E'$ if $X = X'$.
2. All observations are present solely in Ag and X.
3. No agent openness.
4. $|\mathbb{X}_{task}| \geq f$, and $|\mathbb{X}_{agent}| \geq f$.

We define our π lookup table $\pi(G) \rightarrow e \in E$; assumption (1) *allows for this static policy with respect to tasks else we must consider a changing action space despite one observed* X. In Eq. 24, we define the space complexity of tasks as a sum of combinations with replacement. Here $\mathbb{X}_{task}$ is all possible task observations, and $\mathbb{X}_{agent}$ all possible agent observations. Here we use assumptions (2); *the observation is split into our observation of other agents, "agent", and "task" observations that describe the state*, and (3); *the number of agent observations, and the number of agents are fixed.*

$$\pi_{lookup}^{space} = \sum_{i=1}^{|X|} \left(\frac{(|\mathbb{X}_{task}| + i - 1)!}{(|\mathbb{X}_{task}| - 1)! i!} \right) \times |\mathbb{X}_{agent}|^{|Ag|} \tag{24}$$

We find a lower bound only considering $i = |X|$, where the number of tasks present is equal to the largest ever seen number of tasks, from Eq. 24 to Eq. 25. We use assumption (4); *the number of unique task/agent observations are at least their degree of freedom;* to substitute in f and $|Ag|$ making this comparable to our deep RL complexity analysis.

$$\pi_{lookup}^{space} = \Omega\left(\frac{(f \times |X| - 1)!}{(f-1)! |X|!} \times |Ag|^{|Ag|} \right) \tag{25}$$

A.7.2 MOHITO lookup time

With a lookup table, we hash the observation graph which is linear, $\mathcal{O}(|G_{\mathbf{C}}|)$. Then we lookup the preferred $e \in E$ at G which is constant, $\mathcal{O}(1)$. The same operation is performed for critic lookup.

$$\pi_{lookup}^{time} = \mathcal{O}(|G_{\mathbf{C}}|) \tag{26}$$

A.7.3 MOHITO batch size

An ideal batch contains all possible one-step trajectories, (S, A, S'). The range of possible trajectories is bounded when we make the prior lookup table assumptions, Eq. 27, represents the state $S = \{$observations of agents, observations of tasks$\}$ and is squared to account for noisy transitions $S \rightarrow S'$.

$$\text{batch size} = \mathcal{O}\left(|Ag||E| \left(|\mathbb{X}_{agent}|^{|Ag|} \sum_{i=1}^{|X|} \frac{(|\mathbb{X}_{task}| + i - 1)!}{(|\mathbb{X}_{task}| - 1)! i!} \right)^2 \right) \tag{27}$$