GAT-MF: Graph Attention Mean Field for Very Large Scale Multi-Agent Reinforcement Learning

Qianyue Hao
BNRist, Department of Electronic Engineering, Tsinghua University
Beijing, China

Wenzhen Huang
BNRist, Department of Electronic Engineering, Tsinghua University
Beijing, China

Tao Feng
BNRist, Department of Electronic Engineering, Tsinghua University
Beijing, China

Jian Yuan
BNRist, Department of Electronic Engineering, Tsinghua University
Beijing, China

Yong Li*
BNRist, Department of Electronic Engineering, Tsinghua University
Beijing, China

ABSTRACT

Recent advancements in reinforcement learning have witnessed remarkable achievements by intelligent agents ranging from game-playing to industrial applications. Of particular interest is the area of multi-agent reinforcement learning (MARL), which holds significant potential for real-world scenarios. However, typical MARL methods are limited in their ability to handle tens of agents, leaving scenarios with up to hundreds or even thousands of agents almost unexplored. The scaling up of the number of agents presents two primary challenges: (1) agent-agent interactions are crucial in multi-agent systems while the number of interactions grows quadratically with the number of agents, resulting in substantial computational complexity and difficulty in strategies-learning; (2) the strengths of interactions among agents exhibit variations both across agents and over time, making it difficult to precisely model such interactions. In this paper, we propose a novel approach named Graph Attention Mean Field (GAT-MF). By converting agent-agent interactions into interactions between each agent and a weighted mean field, we achieve a substantial reduction in computational complexity. The proposed method offers a precise modeling of interaction dynamics with mathematical proofs of its correctness. Additionally, we design a graph attention mechanism to automatically capture the diverse and time-varying strengths of interactions, ensuring an accurate representation of agent interactions. Through extensive experimentation conducted in both manual and real-world scenarios involving over 3000 agents, we validate the efficacy of our method. The results demonstrate that our method outperforms the best baseline method with a remarkable improvement of 42.7%. Furthermore, our method saves 86.4% training time and 19.2% GPU memory compared to the best baseline method. For reproducibility, our source codes and data are available at https://github.com/tsinghua-fib-lab/Large-Scale-MARL-GATMF.

Corresponding author. Email: liyong07@tsinghua.edu.cn.

CCS CONCEPTS

• **Computing methodologies** → **Multi-agent reinforcement learning**; **Multi-agent systems**.

KEYWORDS

Multi-agent reinforcement learning, large-scale decision problem, mean field, graph attention.

ACM Reference Format:
Qianyue Hao, Wenzhen Huang, Tao Feng, Jian Yuan, and Yong Li*. 2023. GAT-MF: Graph Attention Mean Field for Very Large Scale Multi-Agent Reinforcement Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3580305.3599359

1 INTRODUCTION

In recent years, unprecedented achievements in reinforcement learning (RL) have greatly enhanced human decision-making capabilities in complex situations. RL techniques have found extensive applications in various domains, such as game-playing [42, 54], robotics [4, 43], public health [2, 15, 16], and even nuclear fusion system [11]. The success of RL predominantly centers on single-agent scenarios, while in the real world, systems often comprise multiple agents, and interactions among these agents play a crucial role. Consequently, multi-agent reinforcement learning (MARL) has especially wide applications and developments in corresponding methods are called for.

Previous researchers have done plentiful works on MARL. First, MADDPG [22] outperforms single-agent DDPG in experimental tasks with various goals. Second, in game-playing, a series of studies including QMIX [34], AlphaStar [49] and MAPPO [55] keep surpassing human professional players and achieving remarkable scores on the Starcraft game, a typical MARL benchmark. Further, in social and industrial applications, numerous successful MARL solutions have been proposed, addressing challenges in traffic signal controlling [51], power distribution management [50], cloud computing [1], etc. However, existing studies typically only consider tens of agents, while methods capable for scenarios with up to hundreds or even thousands of agents are almost unexplored.

Despite the prevalence of scenarios involving hundreds or thousands of agents in the real world, there exist two key challenges in scaling up the number of agents in MARL. (1) **Large number of agent-agent interactions**. In multi-agent systems, interactions between agents play a crucial role in the overall system behavior. Therefore, besides simple agent-environment interactions, MARL methods must take the agent-agent interactions into consideration to achieve satisfactory performance. However, in regular algorithms, the number of agent-agent interactions increases following $O(N^2)$ as the number of agents grows to N. The quadratical growth greatly adds to the computational complexity and hinders the agents to learn efficient strategies. (2) **Varying strengths of agent-agent interactions**. Due to the intrinsic dynamics of real-world systems, the strengths of interactions vary not only among each pair of agents but also over time, making it arduous to precisely model these diverse and time-varying interactions. Though we can manually specify the interaction strength of each pair of agents according to prior knowledge of a certain scenario, it requires repetitive manual works when training models to solve problems in different scenarios. On the other hand, it is almost impossible to accomplish such manual works when the number of agents is large.

Facing these challenges, we propose the Graph Attention Mean Field (GAT-MF) method to enable the scaling up of the number of agents in MARL. Firstly, to solve the problem of the unaffordably large number of agent-agent interactions, we develop the previous study of unweighted Mean Field [53] into a weighted version. We mathematically prove the validity of transforming interactions among the agents into the interactions between each agent and a corresponding field, which is obtained through a weighted average over the raw agent-agent interactions. By such conversion, the number of agent-field interactions only scales linearly following $O(N)$ with N agents, alleviating the computational complexity and enhancing the agents' ability to learn effective strategies. Moreover, such conversion preserves the information of different interaction strengths among agents in the weights, which is discarded in the unweighted mean field approach. Secondly, to automatically capture the varying strengths of the interactions, i.e., the weights in calculating the equivalent field, we model the relations among the agents into a graph where each node represents one agent. We introduce a graph attention mechanism to dynamically learn and compute the diverse and time-varying interaction strengths among the agents, obviating the need for prior knowledge or manual efforts in setting interaction strengths. Lastly, we evaluate our GAT-MF method in (1) a grid-world manual scenario with 100 agents and (2) two real-world metropolitan scenario each with more than 3000 agents, which are built according to real-world data (see Section 5). The experimental results demonstrate that the proposed method outperforms existing MARL methods in all scenarios with a performance improvement up to 42.7%, showcasing its capability to scale up to scenarios with a large number of agents. Additionally, our method exhibits high computational efficiency, reducing training time by 86.4% and GPU memory usage by 19.2% compared to the best-performing baseline method.

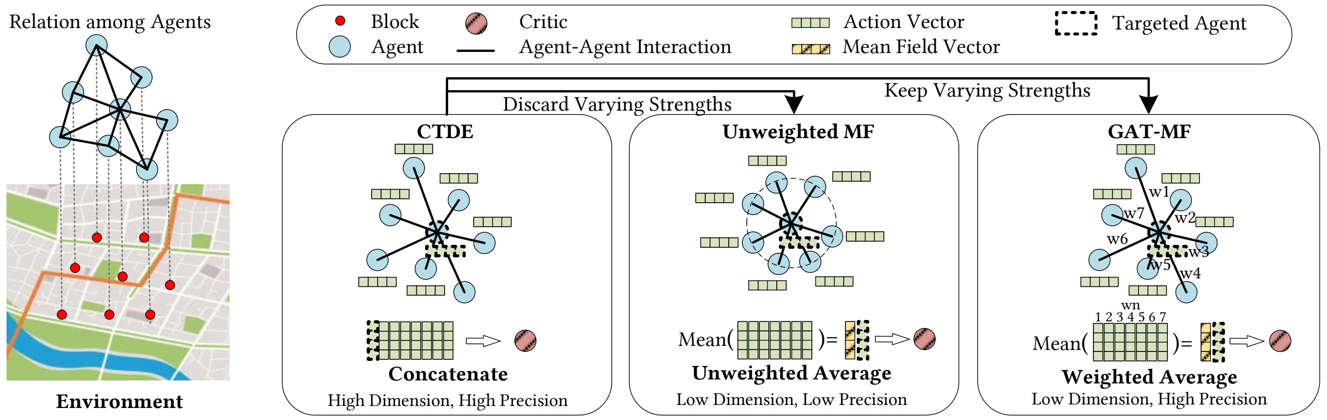
In summary, the main contributions of this work include:

- We develop unweighted Mean Field method into a weighted version and prove its mathematical correctness. This method greatly alleviate the computational complexity in learning efficient strategies and enable it to scale to scenarios with up to thousands of agents.
- We design a graph attention mechanism to automatically capture the varying strengths of the agent-agent interactions, ensuring that our method can precisely model these interactions without prior knowledge of the strengths of the agent-agent interactions in the target scenario.
- We conduct extensive experiments in a manual grid-world scenario and two real-world metropolitan scenarios with more than 3000 agents. The results demonstrate that our method achieves superior performance across scenarios and obtains high computational efficiency comparing existing MARL methods.

2 RELATED WORKS

Large-scale task with RL. One common approach to addressing large-scale task with RL is to aggregate the large number of natural units into a relatively small number of clusters and control each cluster with one agent [15, 32, 51]. Another category of methods decompose the original vast action space into a hierarchical structure based on prior knowledge of the targeted scenario, simplifying the decision-making process [16, 23, 35]. Additionally, some researchers combine human experts' solutions with RL methods to facilitate more efficient strategy learning in large-scale scenarios [15, 19, 33]. Although these works achieve success in their targeted scenarios, the manual techniques of unit aggregation, action space decomposition, or experts' solution collection are largely problem specific and require strong prior knowledge of the scenarios. In contrast, our method provides a direct MARL approach, eliminating the requirement for problem-specific knowledge, thereby offering a more adaptable framework for addressing large-scale RL tasks.

Multi-agent reinforcement learning. There exist abundant works on MARL and here we illustrate the differences among various multi-agent reinforcement learning (MARL) methods with an example in a metropolitan (Figure 1). The main approaches of recently popular MARL solutions can be roughly summarized into several categories, including decentralized training with decentralized execution (DTDE), CTDE, and CTCE. DTDE methods simply trains the agents independently in the environment, such as IQL and IPPO [10]. CTCE means both the training and execution process requires every agent to access global observation, where typical works include HATRPO, HAPPO [18], DGN [17], Mean Field (MF) [53], and communication based methods [26, 26, 29, 37, 46]. On the other hand, CTDE is the most common category of MARL methods. Typically, it includes a centralized critic, which takes in the global observation to estimate the global return; and decentralized actors, which determine the local actions according to local observations. The representations of CTDE methods include MADDPG [22], MAPPO [55], COMA [12] and ATT-MADDPG [25]. Besides, value decomposition (VD) methods is a typical category of CTDE method, which factorizes the joint Q-function into a certain function of the local Q-function of each agent in order to reduce the complexity. For example, VDN [47] uses an additive function, QMIX [34] chooses a monotonic function and QTRAN [44] extends it into a more general function. However, it is not easy to factorize



(a) A multi-agent scenario (b) CTDE method (c) Unweighted MF method (d) Proposed GAT-MF method

Figure 1: Comparison among existing MARL methods and our proposed method. (a) Example of a multi-agent scenario in a metropolitan where each block corresponds to one agent and the relations among adjacent blocks lead to agent-agent interactions. (b) Limitation of the CTDE methods, i.e, high dimension of the concatenated input vector of the critic. (c) Limitation of the unweighted MF, i.e., losing precision on modeling agent-agent interactions using unweighted average. (d) Key design of our proposed GAT-MF method, reducing the dimension while keeping the precision.

the joint Q-function into hundreds or even thousands of terms, limiting the scalability of such methods on a large number of agents. On the other hand, the centralized critic (or value function) requires concatenating the vectors of local states and actions from all agents together as its input to obtain the information on agent-agent interactions, it is obvious that when the number of agents is large, the concatenated vector will be in extremely high dimension, making the training process hard (Figure 1b). This limits these methods to work in scenarios with at most hundreds of agents. In contrast, due to our special design, our proposed method can scale up to scenarios with more than 3000 agents and keeps high computational efficiency. Moreover, there exist works exploring novel paradigms of MARL, such as centralized teacher with decentralized student (CTDS) [56] and personalized training with distilled execution (PTDE) [9].

Mean Field (MF) MARL. In order to reduce the unaffordable high vector dimension in CTDE methods mentioned above and scale up the number of agents, a special technique named Mean Field (MF) [53] is proposed. It approximates the concatenation of actions with the unweighted average of these actions, namely mean field. RL algorithms with MF technique have been successfully applied in industrial scenarios with large number of agents, such as ridesharing order dispatching [20], edge computing [40] and unmanned aerial vehicles (UAVs) controlling [7]. Nevertheless, such approximation regards interactions among all agents equally and discards the varying strengths of these interactions, losing precision in modeling the complex relations among the agents (Figure 1c). In this paper, we develop the MF method into GAT-MF, in which we design a variant of graph attention mechanism to automatically capture the varying strengths of agent-agent interactions and calculate a weighted average of the actions according to their strengths. Therefore, we are able to reduce the vector dimensions, scale up the number of agents, and keep precise modeling of the relations among the agents at the same time (Figure 1d).

Graph model in MARL. There exist previous works modeling the relations among the agents in MARL systems with graphs, in which each agent is represented by a node and each pair of agent-agent interaction is represented by an edge. With such graph, they focus on the local relations among neighboring agents, distilling the agent-agent interactions. One basic approach is considering a given coordination graph from prior knowledge [13, 28] and the action of each agent is obtained considering the influence of its neighbors. Other researchers improve this approach by utilizing graph neural networks (GNNs) to automatically learn the agent-agent interactions rather than obtain them from a given coordination graph. For example, G2ANet [21] combines hard and soft graph attention to indicate whether there exists an interaction between two agents and the importance of the interaction, HAMA [38] designs a hierarchical graph attention network (GAT) to model the hierarchical relationships among multiple agents and MAGIC [30] utilizes GATs to deal with the communications among agents in agent-agent interactions. Furthermore, one theoretical research mathematically shows how compact the agent-agent interactions can be distilled, guiding algorithm designing [31]. Although these works distill the agent-agent interactions by considering neighbors on the graph and simplify the decision-making process, the number of interactions still increases following $O(N^2)$ as the number of agents grows to N, especially when the graph of agents approaches a dense graph. In contrast, we combine GAT mechanism with MF technique and therefore keep the number of interactions at $O(N)$, making it able to scale up to scenarios with a large number of agents.

3 PRELIMINARIES

3.1 Markov Decision Processes (MDPs)

We discuss our method considering a multi-agent version of Markov Decision Processes (MDPs) defined by $\langle n, \mathcal{S}, \rho, \mathcal{A}, P, R, \gamma \rangle$, where n denotes the total number of agents. The global state $\boldsymbol{s} = (s_1, ..., s_n) \in$

$\mathcal{S}$ consists of local state of each agent. The probability distribution of initial state is given by $\rho = \mathcal{D}(\mathcal{S})$, where $\mathcal{D}(\mathcal{S})$ is a collection of probability distribution over the state space $\mathcal{S}$. The joint action $\boldsymbol{a} = (a_1, ..., a_n) \in \mathcal{A}$ consists of local action of each agent and is produced by the policy $\boldsymbol{\pi_\theta} : \mathcal{S} \mapsto \mathcal{D}(\mathcal{A})$, with $\mathcal{D}(\mathcal{A})$ being a collection of probability distribution over the state space $\mathcal{A}$. $\boldsymbol{\pi_\theta}$ is parameterized with $\boldsymbol{\theta} = (\theta_1, .., \theta_n)$ and local action of each agent is produced by $\pi_{\theta_i}(\boldsymbol{s})$. The state transition probability and the one-step reward given the current state and the joint action are defined by $P : \mathcal{S} \times \mathcal{A} \mapsto \mathcal{D}(\mathcal{S})$ and $R : \mathcal{S} \times \mathcal{A} \mapsto \mathbb{R}$ correspondingly. The global one-step reward is the sum of local one-step rewards from each agent, i.e., $R(\boldsymbol{s}, \boldsymbol{a}) = \sum_j r_j(\boldsymbol{s}, \boldsymbol{a})$, and the long-term discounted reward from t_0 is defined by discount factor γ following:

$$\boldsymbol{R_{t_0}} = \sum_{t=t_0}^{T} \gamma^{t-t_0} R(\boldsymbol{s}^t, \boldsymbol{a}^t), \tag{1}$$

where T is the maximum length of an episode and $\gamma \in [0, 1]$.

3.2 Q-Learning and Deep Q-Network (DQN)

Q-Learning [52] is one of the basic RL methods, which learns efficient policy in MDPs through an off-policy manner. It aims at finding a value function $Q^{\boldsymbol{\pi_\theta}}(\boldsymbol{s}, \boldsymbol{a})$ for policy $\boldsymbol{\pi_\theta}$:

$$\begin{aligned} Q^{\boldsymbol{\pi_\theta}}(\boldsymbol{s}, \boldsymbol{a}) &= \mathbb{E}[\boldsymbol{R_t} | \boldsymbol{s} = \boldsymbol{s^t}, \boldsymbol{a} = \boldsymbol{a^t}] \\ &= \mathbb{E}_{\boldsymbol{s'}}[R(\boldsymbol{s}, \boldsymbol{a}) + \gamma \mathbb{E}_{\boldsymbol{a'} \sim \boldsymbol{\pi_\theta}}[Q^{\boldsymbol{\pi_\theta}}(\boldsymbol{s'}, \boldsymbol{a'})]], \end{aligned} \tag{2}$$

where the recursive form is Bellman Equation. In practical training algorithms, this value function is obtained by minimizing the loss function, which is designed with greedy thinking:

$$\mathcal{L} = \mathbb{E}_{(\boldsymbol{s}, \boldsymbol{a}, R, \boldsymbol{s'}) \sim \mathcal{B}}[(Q(\boldsymbol{s}, \boldsymbol{a}) - y)^2], \quad y = R + \gamma \max_{\boldsymbol{a'}} Q'(\boldsymbol{s'}, \boldsymbol{a'}), \tag{3}$$

where $\mathcal{B}$ is the replay buffer collecting experiences $(\boldsymbol{s}, \boldsymbol{a}, R, \boldsymbol{s'})$ from agent-environment interactions and Q' is the target version of Q, whose parameters are synchronized from Q with delay. After obtaining the optimal value function $Q^*(\boldsymbol{s}, \boldsymbol{a})$, the optimal policy $\boldsymbol{\pi}^*$ is obtained in a greedy way:

$$\boldsymbol{\pi}^*(\arg\max_{\boldsymbol{a}} Q^*(\boldsymbol{s}, \boldsymbol{a}) | \boldsymbol{s}) \to 1. \tag{4}$$

Deep Q-Network (DQN) [27] keeps the same mathematical essence as Q-learning but approaches the optimal value function $Q^*(\boldsymbol{s}, \boldsymbol{a})$ with a deep neural network, representing more complex environmental situations and thus can solve problems in more scenarios.

3.3 Policy Gradient (PG) and Deep Deterministic Policy Gradient (DDPG) Algorithm

Since $argmax_{\boldsymbol{a}}$ is used in obtaining the optimal policy in Q-Learning and DQN, they are only practical in scenarios with discrete action space $\mathcal{A}$. To solve problems with continuous action spaces, Policy Gradient (PG) methods are proposed [48]. In a major group of deep PG methods, besides the value function network, the policy $\boldsymbol{\pi_\theta}$ is also approached by a neural network and directly calculates the action $\boldsymbol{a}$ given $\boldsymbol{s}$. These methods keep training the value network in a similar way as Q-Learning but without greed:

$$\mathcal{L} = \mathbb{E}_{(\boldsymbol{s}, \boldsymbol{a}, R, \boldsymbol{s'}) \sim \mathcal{B}}[(Q(\boldsymbol{s}, \boldsymbol{a}) - y)^2], \quad y = R + \gamma Q'(\boldsymbol{s'}, \boldsymbol{a'}), \tag{5}$$

and optimize the policy network to maximize the episode return, following the gradient:

$$\nabla_{\boldsymbol{\theta}} J = \mathbb{E}_{\boldsymbol{a} \sim \boldsymbol{\pi_\theta}}[\nabla_{\boldsymbol{\theta}} \log \boldsymbol{\pi_\theta}(\boldsymbol{a} | \boldsymbol{s}) Q^{\boldsymbol{\pi_\theta}}(\boldsymbol{s}, \boldsymbol{a})]. \tag{6}$$

In practice, similar to the value network Q, the policy network $\boldsymbol{\pi_\theta}$ also has a target copy with delayed parameters synchronization. Deep Deterministic Policy Gradient (DDPG) [41] is a special variant of PG methods where the policy is converted from a probability distribution over the action space to a deterministic action. In the DDPG training process, a small random disturbance is added to the deterministic action, helping the agent explore the potential action space. In this work, we mainly combine our proposed GAT-MF method with the original MADDPG algorithm [22], which is a multi-agent version of DDPG.

3.4 Problem Overview

In this paper, we primarily focus on large-scale multi-agent problems, which typical comprise 10^2 to 10^3 agents. The goal is to find the joint action of the agents at each time step, i.e., $\boldsymbol{a^t}$, and thus to maximize the global return of a whole episode, i.e., $\boldsymbol{R_{t_0}}$, where the global return is decided by the form of the reward function and depends on the specific problem. To achieve this goal, we consider both agent-agent and agent-environment interactions in the system. In these scenarios, the global value function is equal to the sum of local value functions of each agent, i.e., $Q(\boldsymbol{s}, \boldsymbol{a}) = \sum_j Q_j(\boldsymbol{s}, \boldsymbol{a})$. The local value function of agent j, i.e., $Q_j(\boldsymbol{s}, \boldsymbol{a})$, can be split into the sum of $\bar{Q}_j(\boldsymbol{s}, a_j, a_k), k \neq j$, or the sum of $\tilde{Q}_j(s_j, s_k, a_j, s_k), k \neq j$, which represents the local value considering each pair of agent-agent interaction.

4 METHODS

4.1 Overview

As we showed in Section 2 and Figure 1b & c, unweighted MF uses the following approximation on the value function of CTDE when considering agent j:

$$Q_j(\boldsymbol{s}, \boldsymbol{a}) \sim \bar{Q}_j(\boldsymbol{s}, a_j, \bar{a}_j), \quad \bar{a}_j = \frac{1}{|\mathcal{N}^j|} \sum_{k \in \mathcal{N}^j} a_k, \tag{7}$$

where $\mathcal{N}^j$ denotes the neighboring agents of agent j. However, such an unweighted average neglects the fact that the strengths of agent-agent interactions vary among different agent pairs and over time. Therefore, intuitively, we can improve the approximation into a weighted average to maintain such varying strengths. Moreover, besides reducing the dimension of $\boldsymbol{a}$, we can reduce the dimension of $\boldsymbol{s}$ with a similar technique, further reducing the computational complexity. Generally, we propose the following approximation of the value function of CTDE when considering agent j:

$$\begin{aligned} &Q_j(\boldsymbol{s}, \boldsymbol{a}) \sim \tilde{Q}_j(s_j, \tilde{s}_j, a_j, \tilde{a}_j), \\ &\tilde{a}_j = \frac{1}{W_j} \sum_{k \in \mathcal{N}^j} w^{jk} a_k, \ \tilde{s}_j = \frac{1}{W_j} \sum_{k \in \mathcal{N}^j} w^{jk} s_k, \ W_j = \sum_{k \in \mathcal{N}^j} w^{jk}, \end{aligned} \tag{8}$$

where w^{jk} is the weight between agent j and k, reflecting the strength of interaction between them. On the other hand, to apply our method to PG algorithms with actor-critic structures, we design a similar weighted average approximation to reduce the computational complexity of policy function, i.e., the actor, as follow:

$$\pi_j(\boldsymbol{s}) \sim \hat{\pi}_j(s_j, \hat{s}_j), \quad \hat{s}_j = \frac{1}{U_j}\sum_{k\in\mathcal{N}^j} u^{jk} s_k, \quad U_j = \sum_{k\in\mathcal{N}^j} u^{jk}, \tag{9}$$

where u^{jk} is the corresponding weight between agent j and k.

Next, we will mathematically prove why this approximation holds in Section 4.2 and illustrate how we automatically capture the weights $[w^{jk}]$ and $[u^{jk}]$ and implement such approximation in a practical MARL training algorithm in Section 4.3.

4.2 Mathematical Proof

Here we prove why the intuitive approximation of the weighted average on the value function holds.

Theorem 1 (Weighted MF Approximation). *When considering agent j, the centralized value function $Q_j(\boldsymbol{s}, \boldsymbol{a})$ can be approximated by $\tilde{Q}_j(s_j, \tilde{s}_j, a_j, \tilde{a}_j)$.*

Proof. Since

$$\tilde{a}_j = \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk} a_k, \quad W_j = \sum_{k\in\mathcal{N}^j} w^{jk}, \tag{10}$$

we regard each $a_k, k \in \mathcal{N}^j$ as the sum of $\tilde{a}_j$ and a small fluctuation:

$$a_k = \tilde{a}_j + \delta a_{jk}, \tag{11}$$

therefore, we have:

$$\frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}\delta a_{jk} = \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}(a_k - \tilde{a}_j) = \tilde{a}_j - \tilde{a}_j = 0. \tag{12}$$

And similarly, we have:

$$s_k = \tilde{s}_j + \delta s_{jk}, \quad \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}\delta s_{jk} = 0. \tag{13}$$

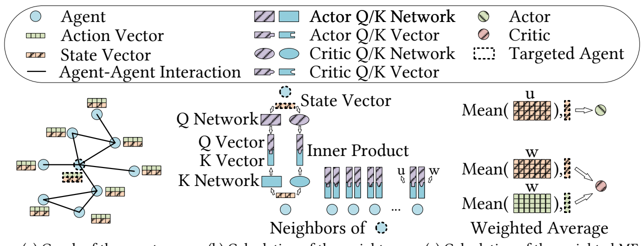
We expand the centralized value function according to Section 3.4, only considering the interactions among neighboring agents:

$$\begin{aligned} Q_j(\boldsymbol{s}, \boldsymbol{a}) &= \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}\tilde{Q}_j(s_j, s_k, a_j, a_k) \\ &= \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}\tilde{Q}_j(s_j, \tilde{s}_j + \delta s_{jk}, a_j, \tilde{a}_j + \delta a_{jk}). \end{aligned} \tag{14}$$

We denote $\tilde{Q}_j(s_j, \tilde{s}_j, a_j, \tilde{a}_j)$ as Q_0 and expand each term in the sum according to Taylor's formula:

$$\begin{aligned} (14) &= \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}[Q_0 + \nabla_{\tilde{s}_j}Q_0 \cdot \delta s_{jk} + \nabla_{\tilde{a}_j}Q_0 \cdot \delta a_{jk} + o^{jk}] \\ &= Q_0 + \nabla_{\tilde{s}_j}Q_0 \cdot \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}\delta s_{jk} \\ &\quad + \nabla_{\tilde{a}_j}Q_0 \cdot \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk}\delta a_{jk} + \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk} o_{jk} \\ &= Q_0 + 0 + 0 + \frac{1}{W_j}\sum_{k\in\mathcal{N}^j} w^{jk} o_{jk} \\ &\approx Q_0 \triangleq \tilde{Q}_j(s_j, \tilde{s}_j, a_j, \tilde{a}_j). \end{aligned} \tag{15}$$

Hence, the centralized value function $Q_j(\boldsymbol{s}, \boldsymbol{a})$ can be approximated with $\tilde{Q}_j(s_j, \tilde{s}_j, a_j, \tilde{a}_j)$ with second order small error. We further proof in Appendix B that the error is bounded within a small symmetric interval $[-CL, CL]$ under the mild condition of the Q-function being L-smooth, where C is a constant. □



(a) Graph of the agents (b) Calculation of the weights (c) Calculation of the weighted MF

Figure 2: Implementation details of the GAT-MF method. (a) Modelling the adjacency among the agents into a graph, where each agent has its state vector and action vector. (b) Process of calculating the weights $[w^{jk}]$, $[u^{jk}]$ through graph attention. (c) Using the obtained weights to calculate the weighted MF vectors to be the inputs of the actor and critic.

4.3 Algorithm Design

Given the correctness of the weighted MF approximation, we illustrate how to implement it into a practical MARL algorithm in Figure 2. We consider scenarios where the agents have fixed relative positions and therefore we can model the adjacency among them into a graph $\mathcal{G}$ (Figure 2a) where each node corresponds to an agent. We consider the neighboring agents of agent j, i.e. $\mathcal{N}^j$, to be the neighbor nodes of agent j. To obtain the weights $[w^{jk}]$ and $[u^{jk}]$, which reflect the varying strengths of interactions among each agent and its neighbors, we design a variant of graph attention (GAT) mechanism to automatically learn them.

In detail, each agent has a pair of query (Q) networks $\mathcal{Q}_a^j, \mathcal{Q}_c^j$ and key (K) networks $\mathcal{K}_a^j, \mathcal{K}_c^j$ with learnable parameters, corresponding to the actor and critic, respectively. In each step, $\mathcal{Q}_a^j, \mathcal{Q}_c^j, \mathcal{K}_a^j, \mathcal{K}_c^j$ take in the current local state vector s_j and produce a pair of Q vectors q_a^j, q_c^j and a pair of K vectors k_a^j, k_c^j, which all have the same dimension. Then, we calculate the weights through the inner product of the corresponding Q and K vectors (Figure 2b), as:

$$u^{jk} = (q_a^j)^T k_a^k, \quad w^{jk} = (q_c^j)^T k_c^k. \tag{16}$$

By this mechanism, we obtain the agent-pair-specific and time-varying weights, reflecting the strengths of agent-agent interactions. Finally, we weigh the state and action vectors by the weights and calculate the MF vectors, which are then used as the inputs of the actor and critic (Figure 2c). The parameters of $\mathcal{Q}_a^j, \mathcal{Q}_c^j, \mathcal{K}_a^j, \mathcal{K}_c^j$ are updated along with the actor and critic networks using the reward signal, and we also apply the technique of target network with delayed parameters synchronization to them. In detail, the parameters of the target networks are updated from the corresponding online network via soft replace with rate κ as:

$$\theta_t \leftarrow \kappa\theta_t + (1-\kappa)\theta_o, \tag{17}$$

Algorithm 1 Multi-agent Training with GAT-MF

```
Require: Number of agents $n$, number of training episodes $M$, model update interval $I$, reward discount factor $\gamma$
Ensure: The trained multi-agent model
 1: Initialize actor networks $\pi$, initialize query (Q) networks $\mathcal{Q}_a, \mathcal{Q}_c$ and key (K) networks $\mathcal{K}_a, \mathcal{K}_c$, the parameters are shared among the agents
 2: Initialize the critic network $Q$
 3: Copy $\pi, \mathcal{Q}_a, \mathcal{Q}_c, \mathcal{K}_a, \mathcal{K}_c$ and get the corresponding target networks $\dot{\pi}, \dot{\mathcal{Q}}_a, \dot{\mathcal{Q}}_c, \dot{\mathcal{K}}_a, \dot{\mathcal{K}}_c$
 4: Copy $Q$ and get the corresponding target network $\dot{Q}$
 5: Initialize the experience replay buffer $\mathcal{B}$
 6: for episode = 1 to $M$ do
 7:     Initialize a random process $\mathcal{P}$ for action exploration
 8:     Initialize the environment and obtain the initial state $\boldsymbol{s} = (s_1, ..., s_n) \in \mathcal{S}$
 9:     for $t = 1$ to max-episode-length do
10:         Calculate the Q vectors and K vectors of each agent: $q_a^j = \mathcal{Q}_a(s_j), q_c^j = \mathcal{Q}_c(s_j), k_a^j = \mathcal{K}_a(s_j), k_c^j = \mathcal{K}_c(s_j)$
11:         for $j = 1$ to $n$ do
12:             Calculate the GAT weights for agent $j$ through the Q vectors and K vectors: $u^{jk} = (q_a^j)^T k_a^k, \quad k \in \mathcal{N}^j$
13:             Calculate the local action with noise through GAT-MF: $a_j = \pi(s_j, \hat{s}_j) + \mathcal{P}$,
                $\hat{s}_j = \frac{1}{U_j} \sum_{k \in \mathcal{N}^j} u^{jk} s_k, U_j = \sum_{k \in \mathcal{N}^j} u^{jk}$
14:         end for
15:         Execute the joint action $\boldsymbol{a} = (a_1, ..., a_n) \in \mathcal{A}$ in the environment, observe the next state $\boldsymbol{s'} = (s'_1, ..., s'_n) \in \mathcal{S}$ and the local one-step rewards $\boldsymbol{r} = (r_1, ..., r_n) \in \mathbb{R}^n$
16:         Store $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ into $\mathcal{B}$
17:         if reach the update interval $I$ then
18:             Sample a random experience $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ from $\mathcal{B}$
19:             Update the network parameters via Algorithm 2
20:         end if
21:     end for
22: end for
23: return The train models for execution $\pi, \mathcal{Q}_a, \mathcal{K}_a$
```

where θ_o denotes the parameters of an online network and θ_t denotes the corresponding target network.

We combine GAT-MF with the original MADDPG algorithm and show the outline of the training algorithm in Algorithm 1. First, we randomly initialize the parameters of online networks and copy them to obtain the corresponding target networks. Considering the homogeneity among the agents, the network parameters are shared among the agents, reducing memory consumption. Second, we perform rollouts in the environment and collect the experiences into the replay buffer. In this step, we calculate the actions through GAT-MF and add random disturbance to the actions for exploration. Third, after collecting a certain number of experiences, we sample experiences from the replay buffer and update the parameters of online actor, critic, query and key networks altogether according to the samples using the reward signal, and then we update the parameters of the corresponding target networks via soft replace.

Algorithm 2 Network Parameters Updating with GAT-MF

```
Require: Number of agents $n$, actor networks $\pi$, query networks $\mathcal{Q}_a, \mathcal{Q}_c$, key networks $\mathcal{K}_a, \mathcal{K}_c$, critic network $Q$ with the corresponding target networks $\dot{\pi}, \dot{\mathcal{Q}}_a, \dot{\mathcal{Q}}_c, \dot{\mathcal{K}}_a, \dot{\mathcal{K}}_c, \dot{Q}$, a sampled experience $(\boldsymbol{s}, \boldsymbol{a}, \boldsymbol{r}, \boldsymbol{s'})$ and soft replace rate $\kappa$
Ensure: The updated networks parameters
 1: for j = 1 to $n$ do
 2:     Set $y_j = r_j + \gamma \dot{Q}(s'_j, \tilde{s}'_j, \dot{a}_j^{s'}, \tilde{\dot{a}}_j^{s'})$
        where $\dot{a}_j^{s'} = \dot{\pi}(s'_j, \hat{s}'_j), \dot{\boldsymbol{a}}^{s'} = (\dot{a}_1^{s'}, ..., \dot{a}_n^{s'})$
        $\hat{s}'_j$ is the GAT-MF vector of $\boldsymbol{s'}$ with weights
        $\dot{q}_a'^j = \dot{\mathcal{Q}}_a(s'_j)$ and $\dot{k}_a'^{\mathcal{N}^j} = \dot{\mathcal{K}}_a(s'_{\mathcal{N}^j})$
        $\tilde{s}'_j$ is the GAT-MF vector of $\boldsymbol{s'}$ with weights
        $\dot{q}_c'^j = \dot{\mathcal{Q}}_c(s'_j)$ and $\dot{k}_c'^{\mathcal{N}^j} = \dot{\mathcal{K}}_c(s'_{\mathcal{N}^j})$
        $\tilde{\dot{a}}_j^{s'}$ is the GAT-MF vector of $\dot{\boldsymbol{a}}^{s'}$ with weights $\dot{q}_c'^j$ and $\dot{k}_c'^{\mathcal{N}^j}$
 3:     Calculate $L_j = (y_j - Q(s_j, \tilde{s}_j, a_j, \tilde{a}_j))^2$
        $\tilde{s}_j$ is the GAT-MF vector of $\boldsymbol{s}$ with weights
        $q_c^j = \mathcal{Q}_c(s_j)$ and $k_c^{\mathcal{N}^j} = \mathcal{K}_c(s_{\mathcal{N}^j})$
        $\tilde{a}_j$ is the GAT-MF vector of $\boldsymbol{a}$ with weights $q_c^j$ and $k_c^{\mathcal{N}^j}$
 4:     Calculate $J_j = Q(s_j, \tilde{s}_j, a_j^s, \tilde{a}_j^s)$ where
        $a_j^s = \pi(s_j, \hat{s}_j), \boldsymbol{a}^s = (a_1^s, ..., a_n^s)$
        $\hat{s}_j$ is the GAT-MF vector of $\boldsymbol{s}$ with weights
        $q_a^j = \mathcal{Q}_a(s_j)$ and $k_a^{\mathcal{N}^j} = \mathcal{K}_a(s_{\mathcal{N}^j})$
        $\tilde{a}_j^s$ is the GAT-MF vector of $\boldsymbol{a}^s$ with weights
        $q_c^j = \mathcal{Q}_c(s_j)$ and $k_c^{\mathcal{N}^j} = \mathcal{K}_c(s_{\mathcal{N}^j})$
 5: end for
 6: Update parameters of $Q, \mathcal{Q}_c, \mathcal{K}_c$, minimizing $L = \sum_j L_j$
 7: Update parameters of $\pi, \mathcal{Q}_a, \mathcal{K}_a$, maximizing $J = \sum_j J_j$
 8: Update $\dot{\pi}, \dot{\mathcal{Q}}_a, \dot{\mathcal{Q}}_c, \dot{\mathcal{K}}_a, \dot{\mathcal{K}}_c$ from parameters of $\pi, \mathcal{Q}_a, \mathcal{Q}_c, \mathcal{K}_a, \mathcal{K}_c$ via soft replace with rate $\kappa$
 9: Update $\dot{Q}$ from parameters of $Q$ via soft replace with rate $\kappa$
10: return The updated networks parameters
```

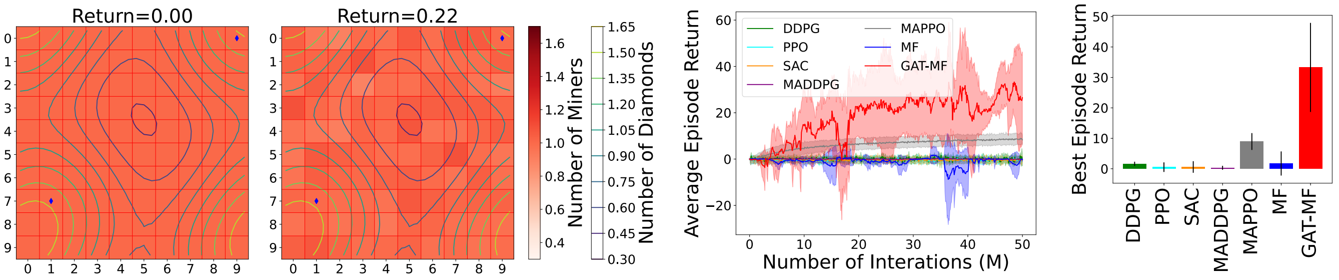
We show the detailed algorithm for parameters updating in Algorithm 2. We repeatedly execute the rollout-update process until convergence and obtain the well-trained GAT-MF models.

We first evaluate our method in a manual grid-world task with 100 agents and provide some straightforward visualizations of the learned policies. We then extend the experiment into real-world metropolitan tasks with more than 3000 agents, fully verifying the ability of our method to scale up to very large number of agents. We show the experimental results in Section 5 and provide detailed hyper-parameters settings used in the experiments in Appendix A. Our experiments are implemented with PyTorch, and the source codes and data for reproducibility are posted at https://github.com/tsinghua-fib-lab/Large-Scale-MARL-GATMF.

5 EXPERIMENTS

5.1 Experimental Settings

Scenario 1: manual grid-world task. First, we start with a diamonds-seeking task in a manual grid world consisting of 10×10 grids with loop boundaries, i.e., going out from the left will loop into the right,



(a) Random initialization (b) Random action, 10 steps (c) Training curves (d) Best episode returns

Figure 3: Experimental settings and results in the grid-world task. (a) An initialized example of the grid world with randomly located diamonds and uniformly distributed miners where each grid corresponds to one agent. (b) The return after 10 steps of random action. (c) Training curves of the proposed GAT-MF, the baselines, and the ablation study with 95%-CI over 5 different random seeds. (d) Best performance of each method with 95%-CI over 5 different random seeds.

and each grid corresponds to one agent. We show a randomly initialized example of the grid world in Figure 3a. Initially, there are two diamond veins buried in two random grids (blue markers in the figure) and the density of diamonds over the world, which is denoted by the contour lines, follows Gaussian distributions centered at the two grids. There are also miners uniformly distributed over the grids, whose number is denoted by the background color. The total number of obtained diamonds is calculated by summing up the product of the number of diamonds and miners at each grid and the return is defined by how many more diamonds are obtained than initialization. In each time step, 10% of the miners from each grid move to one of its neighbors, and the task of the agents is to give the exact numbers of miners to each of the four neighbors to efficiently redistribute the miners and get as many diamonds as possible. We show the distribution of miners and the return after 10 steps of random movements in Figure 3b as an example.

Scenario 2: real-world metropolitan task. In order to verify the ability of our method to scale up to scenarios with a larger number of agents, we conduct further experiment on the real-world task of COVID-19 vaccine allocation. During the pandemic of COVID-19, people in metropolis with high population density and frequent population mobility suffer a high risk of infection [45]. However, vaccines, one of the most important medical resources, are in shortage during the early stage of COVID-19. Therefore, in order to minimize the overall infections, finding an efficient strategy to allocate the limited vaccines to the key population group, e.g., the elders or the health workers, in the metropolis is of vital importance. Previous researchers have built a precise simulator with real-world data to infer the total infections with different vaccine-allocation strategies [8]. We conduct our experiments based on the simulator in two different cities, Atlanta and Miami, ensuring the ability of generalization of our method. In each city, there are thousands of blocks, corresponding to thousands of agents (Figure 4a and 4e). People in each block have different age structures and time-varying patterns visiting points of interest (POI[1]s) in the city. The population mobility leads to contacts among people and complex and time-varying risks of infection. We show the details of the cities in this experiment in Table 1.

[1]Specific locations that someone may find useful or interesting.

Table 1: Details of the cities in experiments.

Name	Atlanta	Miami
Population	7191638	6635035
Number of blocks	3130	3555
Number of POIs	39411	40964
Length of data	1512 hours	

The task of the agents is to find an efficient way to allocate the limited vaccines among the blocks at each time step, reduce the local risk of infection in certain key blocks, and thus minimize the overall infections, which is opposite to the return. We show a more detailed background of this experiment in Appendix C.

5.2 Overall Performance

As we mentioned in Section 4.3, we mainly combine our proposed GAT-MF with the MADDPG algorithm and train the models. We compare the performance of our method with various baseline RL algorithms, which are listed as follows:

- **Global agent methods**: In this group of baselines, we apply widely used single-agent RL methods including **DDPG** [41], **PPO** [39] and **SAC** [14] to the task. We consider the single-agent to be a global one, which takes in the global state $\boldsymbol{s}$ from all grids and gives the joint action $\boldsymbol{a}$ of all grids in the form of a concatenated high dimensional vector.
- **CTDE MARL methods**: In this group of baselines, we apply original versions of **MADDPG** [22] and **MAPPO** [55], which are popular CTDE methods.

Specially, we further design the following ablation study to verify the vital role of our GAT design:

- **Ablation study, w/o GAT**: We substitute the proposed GAT-MF with the unweighted **MF** method as we described in Section 4.1 while keeping the other parts identical.

In the manual grid-world task, we set the local state s_j at grid j to be a five-dimensional vector consisting of the densities of diamonds at the center and the four corners of the grid. The local one-step reward r is set to be the difference in the number of obtained diamonds by the grid and its neighbors before and after the step.

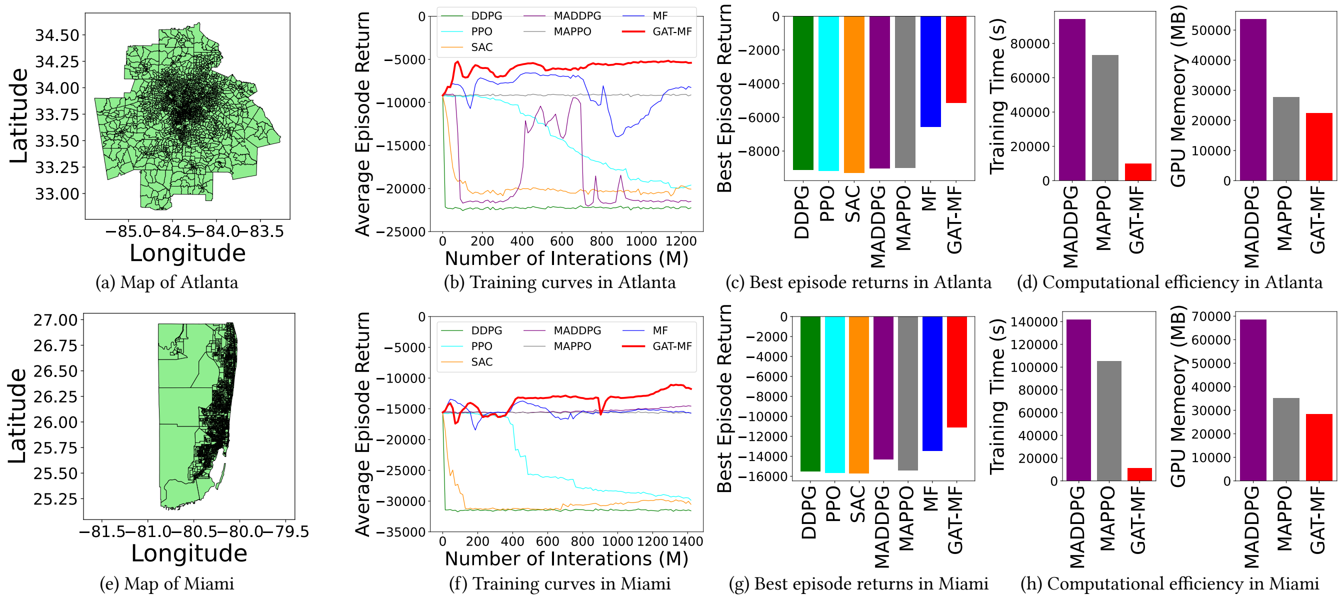


Figure 4: Experimental settings and results in the real-world task. (a) The map of Atlanta, where each block is one unit in the vaccine-allocation task and corresponds to one agent. (b) Training curves of the proposed GAT-MF, the baselines, and the ablation study in Atlanta. (c) Best performance of each method in Atlanta. (d) Training time and GPU memory consumption comparison in Atlanta. (e)-(h) Results of another group of experiments in Miami.

We train the model repeatedly with 5 different random seeds to ensure the robustness of our method. We set 10 steps to be one episode and randomly reinitialize the grid world after each episode during the training.

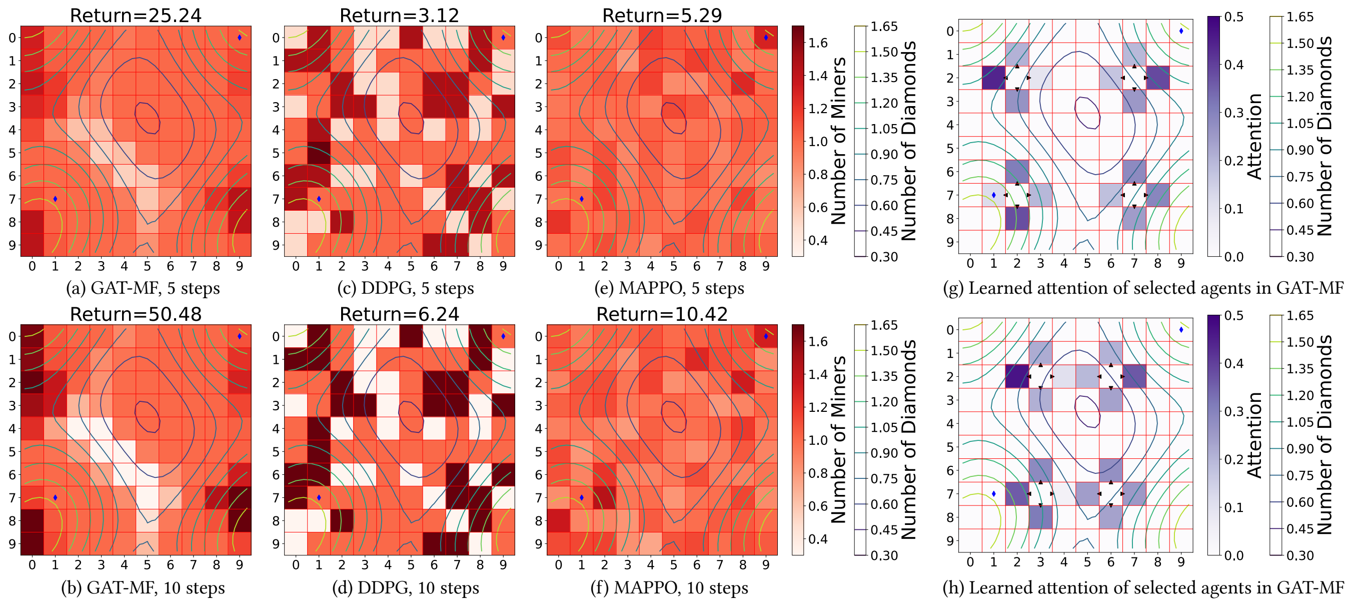
We show the training curves over 50M agent-environment interactions in Figure 3c and extract the best performance of each method in Figure 3d, both together with the 95%-CI over different random seeds. The results illustrate the superior performance of our method with a margin up to 271.9%. Though MAPPO, one of the SOTA MARL methods, is the best baseline, all global agent methods and CTDE MARL methods perform relatively bad due to the high complexity caused by the high dimensional vectors. This stresses the necessity of using the mean-field approximation to reduce the dimension when the number of agents is large. On the other hand, the ablation version of our method, i.e., the MF, suffers sharp performance decay, proving the key role of our GAT design.

In the real-world metropolitan task, the local state s_j at block j consists of the current number of susceptible, infected, and dead people in this block and we set the one-step reward r to be opposite to the number of newly infected people in the step. We set 24 hours as one step during the training and an episode consists of 63 steps (9 weeks). We compare the performance of our method with the same baselines and ablation study, setting the available number of vaccines identical among our method and the baselines.

We show the training curves over more than 1 billion agent-environment interactions in Figure 4b and 4f, and summarize the best performance of each method in Figure 4c and 4g. The results illustrate that in this larger scenario with thousands of agents, all the baseline methods failed to learn useful strategies and some of them are even worse than a randomly initialized model. In contrast, our method reaches a good performance rapidly and converges stably, showing superior performance comparing the best baseline with the margin of 42.7% and 22.9% in Atlanta and Miami, respectively. The unweighted MF method ranks second in both cities, verifying the validity of the mean field approximation. On the other hand, its performance decay comparing the full GAT-MF verifies the key role of our GAT design. In addition, we compare the performance of our GAT-MF method with some well-known graph-based MARL method in Appendix D, again highlighting the superior of out method in large-scale settings. For more precise quantitative understanding, we also provide numerical results for barplots in Figure 3 and Figure 4 in the GitHub repository.

5.3 Visualizations of the Learned Policy

To help understand what the agents learned through training with our method, here we provide some visualizations of the learned policies in the manual grid-world task. In Figure 5a and 5b, we show the distribution of miners after applying the policy from a trained GAT-MF model for 5 and 10 steps. From these we find the agents have learned reasonable policy, quickly making the miners gather at grids with the most diamonds, e.g., the left-upper and left-lower corner in the shown example. On the other hand, we show in Figure 5c-5f the distribution of miners after applying policies from a trained DDPG model and a trained MAPPO model for 5 and 10 steps, which are respective representatives of global agent and CTDE methods. The DDPG policy tends to be like random movements and the MAPPO policy tends to only do very slight movements. We show more visualizations of policies from models



(a) GAT-MF, 5 steps (c) DDPG, 5 steps (e) MAPPO, 5 steps (g) Learned attention of selected agents in GAT-MF
(b) GAT-MF, 10 steps (d) DDPG, 10 steps (f) MAPPO, 10 steps (h) Learned attention of selected agents in GAT-MF

Figure 5: Visualizations of the learned policies in the grid-world task. (a) (b) The distribution of miners after applying the policy from a trained GAT-MF model for 5 and 10 steps. (c) (d) The distribution of miners after applying the policy from a trained DDPG model for 5 and 10 steps. (e) (f) The distribution of miners after applying the policy from a trained MAPPO model for 5 and 10 steps. (g) (h) The learned attentions of some actors at the first step from a trained GAT-MF model. Since one panel cannot hold the attentions of all the actors, we show the attentions of four actors in each panel.

trained via other baseline methods in the GitHub repository and find none of them help the agents learn efficient policy. Once again, we verify the advantage of our method over the existing ones.

We also visualize the learned attentions of some actors at the first step from a trained GAT-MF model in Figure 5g and 5h. We find the attentions vary among the agents and reasonably, the agents tend to pay larger attention to the neighboring agents which are in grids with more diamonds, verifying the validity of the GAT design in capturing the varying strengths of agent-agent interactions. We also show more visualizations of the attentions in the GitHub repository and from the results, we again verify the validity of the GAT design.

5.4 Computational Efficiency

We compare the computational efficiency[2] of our method with the two CTDE methods in the real-world metropolitan task, where there are more than 3000 agents. We show the comparison on training time consumption and GPU memory consumption in Figure 4d and 4h. The results indicate that with the GAT-MF design, which greatly reduces the input dimension of the actor and critic, our method saves 86.4% and 89.2% training time comparing MAPPO when training over the same number of agent-environment interactions in Atlanta and Miami, respectively. Besides, our method saves 19.2% and 19.5% GPU memory comparing MAPPO in the two cities, respectively. In a nutshell, our method works especially well in this scenario where there is a much larger number of agents, not only reaching superior performance but also greatly reducing the training time and GPU memory consumption.

[2]Test on NVIDIA A100 GPU

6 CONCLUSIONS

In this paper, we propose the GAT-MF method, which addresses the challenge of scaling up the number of agents in MARL algorithms. We first intuitively analyzed its advantages over the existing CTDE methods and the unweighted MF method, and then we mathematically proved its correctness. By incorporating the graph attention mechanism, we implemented the theoretical theorem into a practical MARL algorithm. We conducted extensive experiments with the algorithm in both manual and real-world scenarios with more than 3000 agents. The results demonstrate the superior performance and high computational efficiency of our method, showcasing its potential for solving various real-world large-scale problems.

One main limitation of our method is that the GAT mechanism requires the agents to have fixed relative positions and thus the adjacency among them can be modeled as a static graph. However, in some scenarios, the agents are constantly moving and thus do not have fixed relative positions. One possible solution for this is to extend the static graph in this paper into a dynamic one and modify the GAT mechanism to work on dynamic graphs. We leave this as a potential direction for future works.

ACKNOWLEDGMENTS

This work was supported in part by The National Key Research and Development Program of China under grant 2020AAA0106000, the National Natural Science Foundation of China under 61972223, U1936217, 61971267, and Beijing National Research Center for Information Science and Technology.

REFERENCES

[1] Husameldin AM Balla, Chen Guang Sheng, and Weipeng Jing. 2021. Reliability-aware: task scheduling in cloud computing using multi-agent reinforcement learning algorithm and neural fitted Q. *Int. Arab J. Inf. Technol.* 18, 1 (2021), 36–47.

[2] Hamsa Bastani, Kimon Drakopoulos, Vishal Gupta, Ioannis Vlachogiannis, Christos Hadjicristodoulou, Pagona Lagiou, Gkikas Magiorkinis, Dimitrios Paraskevis, and Sotirios Tsiodras. 2021. Efficient and targeted COVID-19 border testing via reinforcement learning. *Nature* 599, 7883 (2021), 108–113.

[3] Wendelin Böhmer, Vitaly Kurin, and Shimon Whiteson. 2020. Deep coordination graphs. In *International Conference on Machine Learning*. PMLR, 980–991.

[4] Lukas Brunke, Melissa Greeff, Adam W Hall, Zhaocong Yuan, Siqi Zhou, Jacopo Panerati, and Angela P Schoellig. 2022. Safe learning in robotics: From learning-based control to safe reinforcement learning. *Annual Review of Control, Robotics, and Autonomous Systems* 5 (2022), 411–444.

[5] Serina Chang, Emma Pierson, Pang Wei Koh, Jaline Gerardin, Beth Redbird, David Grusky, and Jure Leskovec. 2021. Mobility network models of COVID-19 explain inequities and inform reopening. *Nature* 589, 7840 (2021), 82–87.

[6] Serina Chang, Mandy L Wilson, Bryan Lewis, Zakaria Mehrab, Komal K Dudakiya, Emma Pierson, Pang Wei Koh, Jaline Gerardin, Beth Redbird, David Grusky, et al. 2021. Supporting covid-19 policy response with large-scale mobility-based modeling. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2632–2642.

[7] Dezhi Chen, Qi Qi, Zirui Zhuang, Jingyu Wang, Jianxin Liao, and Zhu Han. 2020. Mean field deep reinforcement learning for fair and efficient UAV control. *IEEE Internet of Things Journal* 8, 2 (2020), 813–828.

[8] Lin Chen, Fengli Xu, Zhenyu Han, Kun Tang, Pan Hui, James Evans, and Yong Li. 2022. Strategic COVID-19 vaccine distribution can simultaneously elevate social utility and equity. *Nature Human Behaviour* (2022), 1–12.

[9] Yiqun Chen, Hangyu Mao, Tianle Zhang, Shiguang Wu, Bin Zhang, Jianye Hao, Dong Li, Bin Wang, and Hongxing Chang. 2022. PTDE: Personalized Training with Distillated Execution for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2210.08872* (2022).

[10] Christian Schroeder de Witt, Tarun Gupta, Denys Makoviichuk, Viktor Makoviychuk, Philip HS Torr, Mingfei Sun, and Shimon Whiteson. 2020. Is independent learning all you need in the starcraft multi-agent challenge? *arXiv preprint arXiv:2011.09533* (2020).

[11] Jonas Degrave, Federico Felici, Jonas Buchli, Michael Neunert, Brendan Tracey, Francesco Carpanese, Timo Ewalds, Roland Hafner, Abbas Abdolmaleki, Diego de Las Casas, et al. 2022. Magnetic control of tokamak plasmas through deep reinforcement learning. *Nature* 602, 7897 (2022), 414–419.

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

[13] Carlos Guestrin, Michail G. Lagoudakis, and Ronald Parr. 2002. Coordinated Reinforcement Learning. In *Machine Learning, Proceedings of the Nineteenth International Conference (ICML 2002), University of New South Wales, Sydney, Australia, July 8-12, 2002*, Claude Sammut and Achim G. Hoffmann (Eds.). Morgan Kaufmann, 227–234.

[14] Tuomas Haarnoja, Aurick Zhou, Kristian Hartikainen, George Tucker, Sehoon Ha, Jie Tan, Vikash Kumar, Henry Zhu, Abhishek Gupta, Pieter Abbeel, et al. 2018. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905* (2018).

[15] Qianyue Hao, Wenzhen Huang, Fengli Xu, Kun Tang, and Yong Li. 2022. Reinforcement Learning Enhances the Experts: Large-scale COVID-19 Vaccine Allocation with Multi-factor Contact Network. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. 4684–4694.

[16] Qianyue Hao, Fengli Xu, Lin Chen, Pan Hui, and Yong Li. 2021. Hierarchical Reinforcement Learning for Scarce Medical Resource Allocation with Imperfect Information. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2955–2963.

[17] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. 2018. Graph convolutional reinforcement learning. *arXiv preprint arXiv:1810.09202* (2018).

[18] Jakub Grudzien Kuba, Ruiqing Chen, Muning Wen, Ying Wen, Fanglei Sun, Jun Wang, and Yaodong Yang. 2021. Trust region policy optimisation in multi-agent reinforcement learning. *arXiv preprint arXiv:2109.11251* (2021).

[19] Jingbo Li, Xingjun Zhang, Jia Wei, Zeyu Ji, and Zheng Wei. 2022. GARLSched: Generative adversarial deep reinforcement learning task scheduling optimization for large-scale high performance computing systems. *Future Generation Computer Systems* (2022).

[20] Minne Li, Zhiwei Qin, Yan Jiao, Yaodong Yang, Jun Wang, Chenxi Wang, Guobin Wu, and Jieping Ye. 2019. Efficient ridesharing order dispatching with mean field multi-agent reinforcement learning. In *The world wide web conference*. 983–994.

[21] Yong Liu, Weixun Wang, Yujing Hu, Jianye Hao, Xingguo Chen, and Yang Gao. 2020. Multi-Agent Game Abstraction via Graph Attention Neural Network. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 7211–7218. https://ojs.aaai.org/index.php/AAAI/article/view/6211

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

[23] Yi Ma, Xiaotian Hao, Jianye Hao, Jiawen Lu, Xing Liu, Tong Xialiang, Mingxuan Yuan, Zhigang Li, Jie Tang, and Zhaopeng Meng. 2021. A hierarchical reinforcement learning based optimization framework for large-scale dynamic pickup and delivery problems. *Advances in Neural Information Processing Systems* 34 (2021), 23609–23620.

[24] Hangyu Mao, Wulong Liu, Jianye Hao, Jun Luo, Dong Li, Zhengchao Zhang, Jun Wang, and Zhen Xiao. 2020. Neighborhood cognition consistent multi-agent reinforcement learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 34. 7219–7226.

[25] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, and Zhibo Gong. 2018. Modelling the dynamic joint policy of teammates with attention multi-agent DDPG. *arXiv preprint arXiv:1811.07029* (2018).

[26] Hangyu Mao, Zhengchao Zhang, Zhen Xiao, Zhibo Gong, and Yan Ni. 2020. Learning agent communication under limited bandwidth by message pruning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 5142–5149.

[27] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602* (2013).

[28] Ranjit Nair, Pradeep Varakantham, Milind Tambe, and Makoto Yokoo. 2005. Networked Distributed POMDPs: A Synthesis of Distributed Constraint Optimization and POMDPs. In *Proceedings, The Twentieth National Conference on Artificial Intelligence and the Seventeenth Innovative Applications of Artificial Intelligence Conference, July 9-13, 2005, Pittsburgh, Pennsylvania, USA*, Manuela M. Veloso and Subbarao Kambhampati (Eds.). AAAI Press / The MIT Press, 133–139. http://www.aaai.org/Library/AAAI/2005/aaai05-022.php

[29] Yaru Niu, Rohan R Paleja, and Matthew C Gombolay. 2021. Multi-Agent Graph-Attention Communication and Teaming.. In *AAMAS*. 964–973.

[30] Yaru Niu, Rohan R. Paleja, and Matthew C. Gombolay. 2021. Multi-Agent Graph-Attention Communication and Teaming. In *AAMAS '21: 20th International Conference on Autonomous Agents and Multiagent Systems, Virtual Event, United Kingdom, May 3-7, 2021*, Frank Dignum, Alessio Lomuscio, Ulle Endriss, and Ann Nowé (Eds.). ACM, 964–973. https://doi.org/10.5555/3463952.3464065

[31] Frans A. Oliehoek, Stefan J. Witwicki, and Leslie Pack Kaelbling. 2021. A Sufficient Statistic for Influence in Structured Multiagent Environments. *J. Artif. Intell. Res.* 70 (2021), 789–870. https://doi.org/10.1613/jair.1.12136

[32] Dawei Qiu, Yujian Ye, Dimitrios Papadaskalopoulos, and Goran Strbac. 2021. Scalable coordinated management of peer-to-peer energy trading: A multi-cluster deep reinforcement learning approach. *Applied Energy* 292 (2021), 116940.

[33] Shuhui Qu, Jie Wang, and Juergen Jasperneite. 2019. Dynamic scheduling in modern processing systems using expert-guided distributed reinforcement learning. In *2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 459–466.

[34] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 4295–4304.

[35] Tao Ren, Jianwei Niu, Bin Dai, Xuefeng Liu, Zheyuan Hu, Mingliang Xu, and Mohsen Guizani. 2021. Enabling Efficient Scheduling in Large-Scale UAV-Assisted Mobile-Edge Computing via Hierarchical Reinforcement Learning. *IEEE Internet of Things Journal* 9, 10 (2021), 7095–7109.

[36] Jingqing Ruan, Yali Du, Xuantang Xiong, Dengpeng Xing, Xiyun Li, Linghui Meng, Haifeng Zhang, Jun Wang, and Bo Xu. 2022. GCS: Graph-Based Coordination Strategy for Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2201.06257* (2022).

[37] Heechang Ryu, Hayong Shin, and Jinkyoo Park. 2020. Multi-agent actor-critic with hierarchical graph attention network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 7236–7243.

[38] Heechang Ryu, Hayong Shin, and Jinkyoo Park. 2020. Multi-Agent Actor-Critic with Hierarchical Graph Attention Network. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*. AAAI Press, 7236–7243. https://ojs.aaai.org/index.php/AAAI/article/view/6214

[39] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[40] Dian Shi, Hao Gao, Li Wang, Miao Pan, Zhu Han, and H Vincent Poor. 2020. Mean field game guided deep reinforcement learning for task placement in cooperative multiaccess edge computing. *IEEE Internet of Things Journal* 7, 10 (2020), 9330–9340.

[41] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International*

conference on machine learning. PMLR, 387–395.

[42] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. 2017. Mastering the game of go without human knowledge. *Nature* 550, 7676 (2017), 354–359.

[43] Samarth Sinha, Ajay Mandlekar, and Animesh Garg. 2022. S4RL: Surprisingly simple self-supervision for offline reinforcement learning in robotics. In *Conference on Robot Learning*. PMLR, 907–917.

[44] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 5887–5896.

[45] Andrew J Stier, Marc G Berman, and Luis Bettencourt. 2020. COVID-19 attack rate increases with city size. *arXiv preprint arXiv:2003.10376* (2020).

[46] Sainbayar Sukhbaatar, Rob Fergus, et al. 2016. Learning multiagent communication with backpropagation. *Advances in neural information processing systems* 29 (2016).

[47] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. 2017. Value-decomposition networks for cooperative multi-agent learning. *arXiv preprint arXiv:1706.05296* (2017).

[48] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. 1999. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems* 12 (1999).

[49] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[50] Jianhong Wang, Wangkun Xu, Yunjie Gu, Wenbin Song, and Tim C Green. 2021. Multi-agent reinforcement learning for active voltage control on power distribution networks. *Advances in Neural Information Processing Systems* 34 (2021), 3271–3284.

[51] Tong Wang, Jiahua Cao, and Azhar Hussain. 2021. Adaptive Traffic Signal Control for large-scale scenario with Cooperative Group-based Multi-agent reinforcement learning. *Transportation research part C: emerging technologies* 125 (2021), 103046.

[52] Christopher JCH Watkins and Peter Dayan. 1992. Q-learning. *Machine learning* 8, 3 (1992), 279–292.

[53] Yaodong Yang, Rui Luo, Minne Li, Ming Zhou, Weinan Zhang, and Jun Wang. 2018. Mean field multi-agent reinforcement learning. In *International conference on machine learning*. PMLR, 5571–5580.

[54] Weirui Ye, Shaohuai Liu, Thanard Kurutach, Pieter Abbeel, and Yang Gao. 2021. Mastering atari games with limited data. *Advances in Neural Information Processing Systems* 34 (2021), 25476–25488.

[55] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. 2021. The surprising effectiveness of ppo in cooperative, multi-agent games. *arXiv preprint arXiv:2103.01955* (2021).

[56] Jian Zhao, Xunhan Hu, Mingyu Yang, Wengang Zhou, Jiangcheng Zhu, and Houqiang Li. 2022. CTDS: Centralized Teacher with Decentralized Student for Multi-Agent Reinforcement Learning. *IEEE Transactions on Games* (2022).

A IMPLEMENTATION DETAILS FOR REPRODUCIBILITY

We perform experiments using Python 3.9 and Pytorch 1.11 with NVIDIA GeForce RTX 3090 and NVIDIA A100 GPUs. Here we provide detailed values of the hyper-parameters used in the experiments for reproducibility in Table 2 and Table 3 for the the grid-world task and the real-world task, respectively.

Table 2: Hyper-parameters in the grid-world task.

Hyper-parameter	Value
Slope of the Leaky-ReLU function	0.01
Training episode	100000
Batch size	256
Replay buffer size	3000000
Soft replace rate	0.01
Optimizer	Adam
Learning rate	1×10^{-4}
Weight of noise for exploration	0.1

Table 3: Hyper-parameters in the real-world task.

Hyper-parameter	Value
Slope of the Leaky-ReLU function	0.01
Training episode	110
Batch size	24
Number of parallel rollout	64
Replay buffer size	64*64*1511
Soft replace rate	0.01
Optimizer	Adam
Learning rate	1×10^{-4}
Weight of noise for exploration	0.002

The networks used in our implementations are mainly MLPs and detailed information can be found in our source codes.

B EXTENDED MATHEMATICAL PROOF

Here, we further derive that the approximation error, i.e., the second order remainder o_{jk} in the Taylor's polynomial, is bounded within a small symmetric interval $[-CL, CL]$ under the mild condition of the Q-function being L-smooth, where C is a constant.

Proof. From Taylor's formula, we have

$$\begin{aligned} o_{jk} &= \frac{1}{2}\delta a_{jk}\nabla^2_{\tilde{a}\_j}\tilde{Q}\_j(s_j, s_k, a_j, \tilde{a}\_j)\delta a_{jk} \\ &+ \frac{1}{2}\delta s_{jk}\nabla^2_{\tilde{s}\_j}\tilde{Q}\_j(s_j, \tilde{s}\_j, a_j, a_k)\delta s_{jk}. \end{aligned} \tag{18}$$

First, we consider the first item where we approximate a_k with $\tilde{a}_j$. The following derivation is irrelevant to s_j, s_k, a_j, and for simplicity we denote

$$\tilde{Q}_j(s_j, s_k, a_j, a_k) \triangleq Q(a_k), \quad \tilde{Q}_j(s_j, s_k, a_j, \tilde{a}_j) \triangleq Q(\tilde{a}_j). \tag{19}$$

Suppose that the Q-function is L-smooth, we have

$$\|\nabla Q(a_k) - \nabla Q(\tilde{a}_j)\|_2 \le L\|a_k - \tilde{a}_j\|_2, \tag{20}$$

where $\|\cdot\|_2$ denotes the l_2-norm.

According to the Lagrange's mean value theorem, we have

$$\begin{aligned} \nabla Q(a_k) - \nabla Q(\tilde{a}_j) &= \nabla Q(\tilde{a}_j + (a_k - \tilde{a}_j)) - \nabla Q(\tilde{a}_j) \\ &= \nabla^2 Q(\tilde{a}_j + \epsilon(a_k - \tilde{a}_j))(a_k - \tilde{a}_j), \epsilon \in [0, 1]. \end{aligned} \tag{21}$$

Adding l_2-norm on the both sides of the above equation, from the smoothness condition, we have

$$\begin{aligned} \|\nabla Q(a_k) - \nabla Q(\tilde{a}_j)\|_2 &= \|\nabla^2 Q(\tilde{a}_j + \epsilon(a_k - \tilde{a}_j))(a_k - \tilde{a}_j)\|_2 \\ &\le L\|a_k - \tilde{a}_j\|_2. \end{aligned} \tag{22}$$

We define $\delta a_{jk} = a_k - \tilde{a}_j \triangleq \delta a$ and the normalized vector $\delta\hat{a} \triangleq \frac{\delta a}{\|\delta a\|_2}$ with $\|\delta\hat{a}\|_2 = 1$ and can write the above inequality as

$$\|\nabla^2 Q(\tilde{a}_j + \epsilon\delta a)\delta\hat{a}\|_2 \le L. \tag{23}$$

By arbitrary choice of δa such that the magnitude $\|\delta a\|_2 \to 0$, it follows from above that

$$\|\nabla^2 Q(\tilde{a}_j)\delta\hat{a}\|_2 \le L. \tag{24}$$

By aligning $\delta\hat{a}$ in the direction of the eigenvectors of the Hessian matrix $\nabla^2 Q$, we obtain that for any eigenvalue λ of $\nabla^2 Q$

$$\|\nabla^2 Q(\tilde{a}_j)\delta\hat{a}\|_2 = \|\lambda\delta\hat{a}\|_2 = |\lambda| \cdot \|\delta\hat{a}\|_2 = |\lambda| \le L. \tag{25}$$

As the Hessian matrix $\nabla^2 Q$ is real symmetric and thus diagonalizable, there exist a unit orthogonal matrix U such that

$$U^T(\nabla^2 Q)U = \Lambda \triangleq diag(\lambda_1, \ldots, \lambda_D). \tag{26}$$

Then we have

$$\delta a \cdot \nabla^2 Q \cdot \delta a = (U\delta a)^T \Lambda (U\delta a) = \sum_{i=1}^{D} \lambda_i (U\delta a)_i^2, \tag{27}$$

where

$$-L\|U\delta a\|_2 = -L\sum_{i=1}^{D}(U\delta a)_i^2 \le \sum_{i=1}^{D}\lambda_i(U\delta a)_i^2 \le L\sum_{i=1}^{D}(U\delta a)_i^2 = L\|U\delta a\|_2. \tag{28}$$

Since U is unit orthogonal matrix, we have

$$\|U\delta a\|_2 = \|\delta a\|_2 = (a_k - \tilde{a}_j)^T(a_k - \tilde{a}_j) = \|a_k\|_2 + \|\tilde{a}_j\|_2 - 2a_k^T\tilde{a}_j \le C, \tag{29}$$

where C is a constant bound related to the magnitude of action vectors.

Put the above together, we have

$$-\frac{1}{2}CL \le \frac{1}{2}\delta a \cdot \nabla^2 Q \cdot \delta a \le \frac{1}{2}CL. \tag{30}$$

Similarly, we can prove that the second item in o_{jk} also follows

$$-\frac{1}{2}CL \le \frac{1}{2}\delta s \cdot \nabla^2 Q \cdot \delta s \le \frac{1}{2}CL. \tag{31}$$

Therefore, we have proved that the approximation error, i.e., the second order remainder, is bounded as

$$-CL \leq o_{jk} \leq CL. \tag{32}$$

□

Specifically in the implementation, the constant C can be controlled to a relatively small number by normalization on the state vector s and action vector a. Also, Q-function tends to be smooth in most scenarios, especially those with continuous state and action spaces, and thus the constant L can be considered small. Put together, the approximation error in our paper is bounded within a small symmetric interval, indicating that our GAT-MF approach can approximate the exact Q-function precisely.

C BACKGROUND OF THE REAL-WORLD METROPOLITAN TASK

In this section, we introduce more details of the background of the real-world vaccines-allocation experiment. The long-lasting global pandemic spreading of COVID-19 has caused countless damage to the social economy and people's daily life. With the aim of better understanding the pandemic spreading dynamics, there exist many works on modeling and simulating the COVID-19 pandemic spreading with various mathematical models and data sources [5, 6, 8]. In this paper, we mainly conduct our experiments based on the Behavior and Demography informed epidemic model (BD model) [8], which is an improved version of the meta-population model [5].

In general, the BD model simulates the pandemic spreading inside the metropolitan statistical areas (MSA[3]s), with the smallest unit of census block groups (CBG[4]s) and points of interest (POI[5]s). Typically, there are thousands of CBGs and more than ten thousand of POIs within one MSA. The BD model divides the pandemic spreading process into two parts, i.e., Intra-CBG transmissions Inter-CBG transmissions, and models them respectively. For the Intra-CBG transmissions, it maintains a local Susceptible-Exposed-Infected-Removed (SEIR) model in each CBG, where S denotes susceptible, E denotes exposed, I denotes infected and R denotes removed. There is a proportion of infected cases become reported cases and the proportion is decided by the testing capability. Only reported cases are observable while the rest of the infected cases are not observable. There are a proportion of removed cases turn out to be deaths according to the infection-fatality rate (IFR) in each CBG while others turn out to recover. As is known to us, the death rate is strongly related to age and thus the IFR is estimated by the population age structure of each CBG and age-specified death risks. On the other hand, the BD-model mainly models the Inter-CBG transmissions part based on the population mobility and contact network among CBGs and POIs. Inter-CBG transmissions happen when susceptible individuals from a certain CBG encounter infected individuals from another CBG when visiting a POI. The transmission probability varies among the POIs, which is positively related to people's average dwelling time at a certain POI and is inversely proportional to the POI's floor space, reflecting the population density in such POI. The effect of vaccines on the transmission process is modeled as an equivalent reduction in the infection rate, which means the probability for susceptible individuals in a certain CBG to be infected and turn into an exposed case in contact with infected individuals reduces proportionally to the percentage of vaccinated individuals in such CBG. We assume that an individual will obtain 100% immunity after one dose of vaccine injection for simplification in this paper while the real-world situations of two-injection vaccines and < 100% immunity can be modeled with no essential difference by simply changing the parameters.

The required data of population mobility, i.e., how people from each CBG visit each POI and encounter people from other CBGs, are originally captured by previous researchers from the SafeGraph open data[6] and are available online[7] [5]. In the study of BD-model, the researchers further process the data into a suitable form to feed into the simulator. They have verified the accuracy of the BD model by comparing the simulation results by computers with the reported real-world situations. Results show that the model can reflect the real-world pandemic spreading situations precisely and thus we can conduct experiments based on it with confidence.

D EXTENDED BASELINE COMPARISONS

In table 4, we provide extended comparisons with some well-known graph-based MARL method in our real-world metropolitan task with thousands of agents to highlight that our GAT-MF is superior for large-scale MARL settings.

- **DGN** [17]: This work applied GCNs to aggregate local observations of all agents and used the aggregated vector as the input of Q-function.
- **DCG** [3]: This work factorized the Q-function according to the coordination graph (CG), into the sum of "utility functions" of each node (agent) and "payoff function" of each edge (agent-agent interaction).
- **GCS** [36]: This work generated directed acyclic graphs (DAGs) with GAT encoders and used the DAGs as action coordination graph (ACG). Based on the ACG, they obtained coordinated actions for agents.
- **NCC** [24]: This work obtained inspiration from psychology and sociology and aimed to keep the 'cognition' among neighboring agents consistent to ensure better cooperation. They designed GCNs and variational autoencoders to capture the 'cognition' of each agent from the observations and used KL-divergence to keep the neighborhood cognition consistency.

Table 4: Best episode return in the real-world metropolitan task

	Atlanta	Miami
DGN	-8968.002 (-73.727%)	-15398.495 (-38.969%)
DCG	-6110.560 (-18.373%)	-13040.075 (-17.684%)
GCS	-7316.499 (-41.734%)	-15654.693 (-41.281%)
NCC	The complexity of NCC is too high to test	
Ours	-5162.126	-11080.564

[3]Regions with a relatively high population density and close economic ties throughout the area.
[4]The smallest geographical unit for which the bureau publishes sample data.
[5]Specific locations that someone may find useful or interesting.

[6]https://www.safegraph.com/
[7]https://covid-mobility.stanford.edu//datasets/