Attention Cooperative Task Offloading and Service Caching in Edge Computing

Zhixiu Yao, *Member, IEEE*, Yun Li, *Member, IEEE*, Shichao Xia, Guangfu Wu
School of Communication and Information Engineering,
Chongqing University of Posts and Telecommunications, Chongqing 400065, China

GLOBECOM 2022 - 2022 IEEE Global Communications Conference | 978-1-6654-3540-6/22/$31.00 ©2022 IEEE | DOI: 10.1109/GLOBECOM48099.2022.10001202

***Abstract*—Mobile edge computing (MEC) enables various services to be cached in close proximity to the user equipments (UEs), thereby reducing the computing delay of many emerging applications. Nevertheless, The limited storage capacity of edge servers requires judicious design of service caching as well as task offloading to maximize edge computing performances. In this paper, we formulate a cooperative task offloading, service caching, and transmit power allocation problem to minimize the cost of computing delay and energy consumption of UEs. To address this problem, we propose a graph attention based multi-agent deep deterministic policy gradient (GAT-MADDPG) algorithm, in which a multi-headed graph attention mechanism is incorporated into the centralized critic network to learn the attentive cooperation policies. Simulation results show that the proposed GAT-MADDPG algorithm exhibits an effective performance improvement.**

***Index Terms*—Task offloading, service caching, multi-agent reinforcement learning, graph attention network.**

I. Introduction

With the advancement of the Internet of Things (IoT) and mobile communication technologies, various computation-intensive and delay-sensitive applications (e.g., smartphones, augmented reality, and interactive online gaming) are growing exponentially. Such massive applications with heterogeneous service requirements have brought urgent challenges to the traditional computing models [1].

In recent years, integrating mobile edge computing (MEC) with ultra-dense cellular network has been expected as an effective solution to support the ubiquitous ultra-low latency computing service. Specifically, MEC server pushes the cloud computing capabilities to the small cell network, which enables various services to be cached close to user equipments (UEs). Service caching refers to caching the related data (e.g., libraries and databases) in MEC servers for executing computing tasks. Nevertheless, due to the limited storage resource of an MEC server, only a small set of popular services can be cached at the same time [2]. Therefore, the efficient service caching mechanism is urgently needed for the resource-constrained MEC network, and it is in general a complicated problem tightly coupled with task offloading decisions.

Existing efforts [3], [4], [5] have addressed task offloading and service caching issues based on traditional optimization methods that require complete information in the network. However, it is usually not available in the practical dynamic and decentralized deployment MEC networks. A few recent studies [6], [7] have attempted deep reinforcement learning (DRL) approaches to address the problem of incomplete information in dynamic MEC network. In [6], the authors proposed a centralized DRL algorithm to minimize the task computing delay and energy consumption cost by jointly optimizing the task offloading and service caching. Nevertheless, the centralized DRL methods require global state information that is difficult to obtain in the decentralized MEC network. Multi-agent reinforcement learning (MARL) is an effective approach that can realize distributed learning for conducting a collaborative task. Inspired by this, the authors in [7] developed a distributed MARL-based edge caching network, in which the agents can learn cooperative polices to achieve higher rewards.

The above mentioned works, however, do not explicitly capture any underlying structure present in MEC network. For example, since the neighboring MEC servers usually have the same popular service requests, MEC servers that are closer together have greater influence on each other's caching decision [8], [9]. Although MARL approaches can learn cooperative polices, the agents can not distinguish the crucial information before making decisions, which results in poor performances due to the interference from worthless information of other agents [10], [11]. To this end, we develop a cooperative task offloading and service caching scheme based on the graph attention multi-agent reinforcement learning. The main contributions of this paper are summarized as follows.

- We formulate a cooperative task offloading, service caching, and transmit power allocation optimization problem for the MEC-enabled small cell network, which aims to minimize the long-term average cost of computing delay and energy consumption of UEs in the network.
- We propose a graph attention based multi-agent deep deterministic policy gradient (GAT-MADDPG) algorithm to solve the formulated problem. A multi-head graph attention mechanism is incorporated into the centralized critic network, which enables agents selectively attend to information of other agents.
- We conduct numerical simulations to evaluate the effectiveness of the proposed algorithm by considering performance metrics such as cache hit rate and cost.

The rest of this paper is organized as follows. Section II describes the system model. The problem formulation is given in Section III. Section IV introduces the detailed designs of the

proposed GAT-MADDPG algorithm. The experimental results and analysis are presented in Section V. Finally, Section VI concludes the paper.

II. SYSTEM MODEL

We consider a network with MEC servers equipped at multiple small cell base stations (BSs) as illustrated in Fig. 1. Let $\mathcal{N} = \{1, 2, \cdots, N\}$ be the set of BSs within the network, where $N = |\mathcal{N}|$ is the number of BSs. Denote $\mathcal{M}^j = \left\{1, 2, \cdots, M^{(j)}\right\}$ as the set of UEs under the coverage of BS j, where $M^{(j)} = \left|\mathcal{M}^j\right|$ is the number of UEs in BS j. For the UE $i^{(j)} \in \mathcal{M}^j$, the BS j is called local BS. In this paper, we adopt the time-slotted dynamic MEC network, where the set of time slots is denoted by $\mathcal{T} = \{1, 2, \cdots\}$. The time slot is divided with an identical duration, which is denoted as $t \in \mathcal{T}$.

A. Computing Task Model

Each UE periodically generates tasks with different computation delay and service requirements. Once a computing task with a particular service is generated, the associated data (e.g., libraries and databases) with this service requires to be configured to process the task. Let $\mathcal{K} = \{1, 2, \cdots, K\}$ be the set of service catalog in the network, where $K = |\mathcal{K}|$ is the number of service types. Then the computing task generated by UE $i^{(j)}$ at time slot t can be characterized by four parameters $\varphi_{i^{(j)}}(t) = \{d_{i^{(j)}}(t), k_{i^{(j)}}(t), \rho_{i^{(j)}}(t), \tau_{i^{(j)}}(t)\}$, where $d_{i^{(j)}}(t)$ is the data size, $k_{i^{(j)}}(t) \in \mathcal{K}$ is the required service, $\rho_{i^{(j)}}(t)$ is the required CPU cycles to execute one-bit input, and $\tau_{i^{(j)}}(t)$ is the maximum delay tolerance. Then all UEs' task generated under the radio coverage of BS j can be expressed as $\boldsymbol{\varphi}_j(t) = \left\{\varphi_{1^{(j)}}(t), \varphi_{2^{(j)}}(t), \cdots, \varphi_{M^{(j)}}(t)\right\}$. The UE $i^{(j)}$ periodically sends a resource access request with the detailed information about the generated computing task to its local BS j, and then the task offloading and resource allocation policies are determined by the local BS j according to UE $i^{(j)}$'s quality of experience (QoE).

B. Edge Service Caching Model

Each BS can place specific computing services, which allows computing tasks with the services to be processed on the BS to reduce the offloading delay. However, not all the services can be cached at a BS with limited storage resource. Denote the storage space of service $k \in \mathcal{K}$ as c_k, and the storage capacity of BS j as C_j. Moreover, let binary variable $\beta_{j,k}(t) \in \{0, 1\}$ be the service caching decision of service k in BS j at time slot t, where $\beta_{j,k}(t) = 1$ if service k is cached at BS j, and $\beta_{j,k}(t) = 0$ otherwise. Accordingly, the services caching decision of BS j at time slot t can be expressed as $\boldsymbol{\beta}_j(t) = \{\beta_{j,1}(t), \beta_{j,2}(t), \cdots, \beta_{j,K}(t)\}$. Besides, we have the constraint $\sum_{k \in \mathcal{K}} \beta_{j,k}(t) c_k \leq C_j$ since the total cached services cannot exceed BS j's storage capacity. Moreover, assume that all BSs in the network can exchange services caching information with each other, and the remote cloud has sufficient computing and storage resources to store all the services.

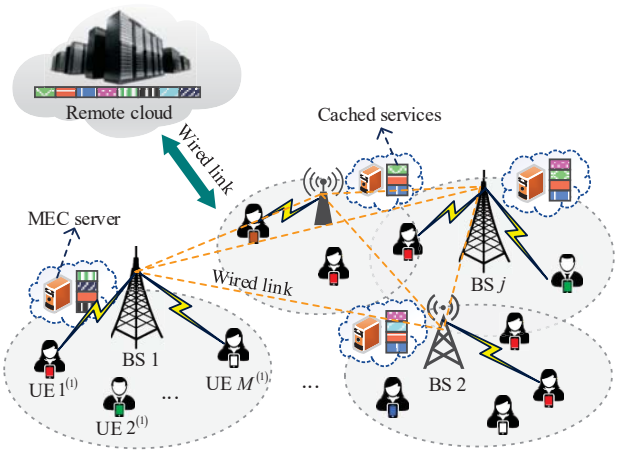

Fig. 1: System Model.

C. Computing Task Offloading Model

In this paper, we consider BSs cooperatively provide computing services to UEs. Specifically, there are three offloading policies for each UE: 1) the task of the UE can be executed at the MEC server co-located with local BS, 2) the task is offloaded to another BS for execution through local BS, or 3) the task is sent to the remote cloud center through local BS. Let $b_{i^{(j)}}(t) \in \{0\} \cup \mathcal{N}$ be the task offloading decision of UE $i^{(j)}$ at time slot t, where $b_{i^{(j)}}(t) = 0$ if UE $i^{(j)}$ offload computing task to local BS j and BS j sends the task to remote cloud for execution, and $b_{i^{(j)}}(t) = j$ if UE $i^{(j)}$'s computing task is processed by its local BS j. $b_{i^{(j)}}(t) = j'$ ($j' \in \mathcal{N} \backslash \{j\}$) if the computing task of UE $i^{(j)}$ is send to local BS j and BS j sends the task to another BS in the network for execution. Then the task offloading decisions of all UEs in BS j at time slot t can be expressed as $\boldsymbol{b}_j(t) = \{b_{1^{(j)}}(t), b_{2^{(j)}}(t), \cdots, b_{M^{(j)}}(t)\}$.

1) Local Server Offloading Model: Once a UE offloads task to its local BS, the task will be uploaded to the BS first and then processed at edge server co-located with the BS. After performing the task, the local BS returns computing results to the UE. In this paper, we mainly focus on the uplink transmission delay and task computation delay. Since the small data size of computing result, the transmission delay of the result is ignored [12]. We consider the uplink transmission model between UEs and BS adopts orthogonal frequency division multiple access. Let $g_{i^{(j)},j}$ be the channel gain between UE $i^{(j)}$ and BS j, then the achieved uplink transmission rate from UE $i^{(j)}$ to BS j at time slot t can be computed by

$$r_{i^{(j)},j}(t) = w \log_2\left(1 + \frac{p_{i^{(j)}}(t)\, g_{i^{(j)},j}}{N_0}\right), \tag{1}$$

where w is the channel bandwidth, $p_{i^{(j)}}(t)$ is the transmit power of UE $i^{(j)}$ at time slot t, and N_0 is the noise power.

Accordingly, the uplink transmission delay from UE $i^{(j)}$ to BS j at time slot t is computed by

$$T^{\text{tran}}_{i^{(j)},j}(t) = \frac{d_{i^{(j)}}(t)}{r_{i^{(j)},j}(t)}. \tag{2}$$

The BSs with related services are responsible for the computing task processing. Denote $f_j(t)$ as the CPU frequency of the BS j, then the task computation delay of UE $i^{(j)}$ at time slot t can be expressed as

$$T^{\text{comp}}_{i^{(j)},j}(t) = \frac{d_{i^{(j)}}(t)\,\rho_{i^{(j)}}(t)}{f_j(t)}. \tag{3}$$

Based on the above assumptions, we can obtain the total offloading delay of UE $i^{(j)}$'s computing task when processed by local BS j at time slot t as follows

$$T_{i^{(j)},j}(t) = T^{\text{tran}}_{i^{(j)},j}(t) + T^{\text{comp}}_{i^{(j)},j}(t). \tag{4}$$

2) Cooperative Offloading Model: In this paper, we consider the computing task is processed in a cooperative manner. Specifically, if the computing task is processed cooperatively, the computing task will be sent to another BS or the remote cloud by the local BS. Let $r_{j,j'}(t)$ and $r_{j,0}(t)$ be the average transmission rate from BS j to BS j' ($j' \in \mathcal{N}\backslash\{j\}$) and remote cloud, respectively. Then the total transmission delay from UE $i^{(j)}$ to BS j and to the remote cloud at time slot t are calculated, respectively, as

$$T^{\text{tran}}_{i^{(j)},j'}(t) = \frac{d_{i^{(j)}}(t)}{r_{i^{(j)},j}(t)} + \frac{d_{i^{(j)}}(t)}{r_{j,j'}(t)}, \tag{5}$$

$$T^{\text{tran}}_{i^{(j)},0}(t) = \frac{d_{i^{(j)}}(t)}{r_{i^{(j)},j}(t)} + \frac{d_{i^{(j)}}(t)}{r_{j,0}(t)}. \tag{6}$$

Let $f_{j'}(t)$ be the CPU frequency of the BS j', then task computation delay at time slot t can be computed by

$$T^{\text{comp}}_{i^{(j)},j'}(t) = \frac{d_{i^{(j)}}(t)\,\rho_{i^{(j)}}(t)}{f_{j'}(t)}. \tag{7}$$

Furthermore, we ignore the computation delay of the remote cloud since it has sufficient computing resource. We can obtain the offloading delay of UE $i^{(j)}$'s computing task when processed by BS j' and remote cloud at time slot t, respectively, as

$$T_{i^{(j)},j'}(t) = T^{\text{tran}}_{i^{(j)},j'}(t) + T^{\text{comp}}_{i^{(j)},j'}(t), \tag{8}$$

$$T_{i^{(j)},0}(t) = T^{\text{tran}}_{i^{(j)},0}(t). \tag{9}$$

3) Energy Consumption Model: Based on the above analysis, the energy consumption of UE $i^{(j)}$ only occurs in the process of transmitting the computing task to its local BS j, which can be computed by

$$E_{i^{(j)}}(t) = p_{i^{(j)}}(t)\,T^{\text{tran}}_{i^{(j)},j}(t). \tag{10}$$

III. PROBLEM FORMULATION

Given the task offloading, transmit power allocation, and service caching decisions, the task offloading delay of UE $i^{(j)}$ at time slot t can be calculated as

$$T_{i^{(j)}}(t) = \begin{cases} T_{i^{(j)},j}(t), & \text{if } b_{i^{(j)}}(t) = j, \beta_{j,k_{i^{(j)}}(t)}(t) = 1 \\ T_{i^{(j)},j'}(t), & \text{if } b_{i^{(j)}}(t) = j', \beta_{j',k_{i^{(j)}}(t)}(t) = 1 \\ T_{i^{(j)},0}(t), & \text{if } b_{i^{(j)}}(t) = 0. \end{cases} \tag{11}$$

Then we define the computing task offloading cost of UEs $i^{(j)}$ at time slot t as

$$Cost_{i^{(j)}}(t) = \omega_d T_{i^{(j)}}(t) + \omega_e E_{i^{(j)}}(t), \tag{12}$$

where ω_d and ω_e are the weights of task offloading delay and energy consumption, respectively.

The objective of this paper is to make cooperative task offloading, transmit power control, and service caching decisions to minimize the long-term average cost of all UEs in the network. We then formulate the optimization problem as follows

$$\min_{\substack{\boldsymbol{\beta}(t),\boldsymbol{b}(t),\\ \boldsymbol{p}(t),\forall t}} \quad \lim_{|\mathcal{T}|\to\infty} \frac{1}{|\mathcal{T}|} \sum_{t\in\mathcal{T}} \sum_{j\in\mathcal{N}} \sum_{i^{(j)}\in\mathcal{M}^j} Cost_{i^{(j)}}(t) \tag{13}$$

$$\text{s.t.} \quad b_{i^{(j)}}(t) \in \{0\} \cup \mathcal{N}, i^{(j)} \in \mathcal{M}^j, j \in \mathcal{N}, \tag{13a}$$

$$\beta_{j,k}(t) \in \{0,1\}, k \in \mathcal{K}, j \in \mathcal{N}, \tag{13b}$$

$$T_{i^{(j)}}(t) \le \tau_{i^{(j)}}(t), i^{(j)} \in \mathcal{M}^j, j \in \mathcal{N}, \tag{13c}$$

$$\sum\nolimits_{k\in\mathcal{K}} \beta_{j,k}(t)\, c_k \le C_j, k \in \mathcal{K}, j \in \mathcal{N}, \tag{13d}$$

where $\boldsymbol{\beta}(t) = \{\boldsymbol{\beta}_1(t), \cdots, \boldsymbol{\beta}_N(t)\}$ is the set of service caching decisions, $\boldsymbol{b}(t) = \{\boldsymbol{b}_1(t), \cdots, \boldsymbol{b}_N(t)\}$ is the set of offloading decisions, and $\boldsymbol{p}(t) = \{\boldsymbol{p}_1(t), \cdots, \boldsymbol{p}_N(t)\}$ is the set of transmit power allocation decisions and $\boldsymbol{p}_j(t) = \{p_{1^{(j)}}(t), p_{2^{(j)}}(t), \cdots, p_{M^{(j)}}(t)\}$. Constraint (13c) indicates the offloading delay of a task can not exceed the maximum delay tolerance, and constraint (13d) specifies that the cached services cannot exceed the storage capacity at each BS.

The optimal solution to the above problem requires complete information about the requested computing task in the network. However, it is in general not available in the practical decentralized MEC network. Moreover, the problem (13) is a mixed integer nonlinear programming problem, which is an NP-hard problem and difficult to be solved by the traditional optimization methods even with the complete information. Thus, we transform the problem (13) into a decentralized partially observable Markov decision process (Dec-POMDP). And a multi-agent reinforcement learning algorithm is leveraged to solve the Dec-POMDP problem, where each BS is an agent to learn the optimal solutions in a cooperative manner. Besides, to learn better cooperation information, we integrate graph neural network (GNN) with MARL by embedding the graph-structured information of agents into the training process.

IV. GAT-MADDPG BASED SOLUTION FOR TASK OFFLOADING AND SERVICE CACHING

In this section, we propose a graph attention based multi-agent deep deterministic policy gradient (GAT-MADDPG) algorithm to address the problem (13). Specifically, we first transform the problem (13) into a Dec-POMDP. Then, the details of the GAT-MADDPG solution are presented.

A. Problem Transformation

We transform the formulated problem into a Dec-POMDP for N agents. In the following, we refer to BS j as an

agent j. There is a tuple $(\mathcal{S}, \mathcal{O}, \mathcal{A}, \mathcal{R})$ to describe the interaction process in Dec-PODMP, where $\mathcal{S}$ describes the global state space and the environment is in global state $s(t) \in \mathcal{S}$ at time slot t, $\mathcal{O} = \{\mathcal{O}_1, \mathcal{O}_2, \cdots, \mathcal{O}_N\}$ is the set of observations, $\mathcal{A} = \{\mathcal{A}_1, \mathcal{A}_2, \cdots, \mathcal{A}_N\}$ is the set of actions, and $\mathcal{R} = \{\mathcal{R}_1, \mathcal{R}_2, \cdots, \mathcal{R}_N\}$ is the set of rewards. Each agent j receives a local observation $o_j(t) \in \mathcal{O}_j$, takes an action $a_j(t) \in \mathcal{A}_j$, and obtains an individual reward $r_j(t) \in \mathcal{R}_j$. The ultimate goal is to maximize the sum of the expected rewards of all agents.

1) Environment State: At the beginning of the time slot t, the agents receive the detailed task information (i.e., the task size, required services, required CPU cycles, and delay tolerance) from UEs within their coverage area. Then the task offloading, transmit power allocation, and service caching actions are determined by agents. Thus, the environment state can be described as $s(t) = \{\boldsymbol{\varphi}_1(t), \boldsymbol{\varphi}_2(t), \cdots, \boldsymbol{\varphi}_N(t)\}$.

2) Observation: The local observations of each agent j at time slot t can be described as the received computing task information, which is $o_j(t) = \boldsymbol{\varphi}_j(t)$.

3) Action: Given the current policy and the corresponding observation, each agent j takes action from its action space. The action of agent j at time slot t can be described as $a_j(t) = \{\boldsymbol{\beta}_j(t), \boldsymbol{b}_j(t), \boldsymbol{p}_j(t)\}$.

4) Reward: According to the problem (13), each agent aims to minimize the overall cost of its associated UEs while satisfying the delay requirements. Therefore, the reward of agent j can be calculated by

$$r_j(t) = U_j(t) + \sum_{i^{(j)} \in \mathcal{M}^j} \left(Y_{i^{(j)}}(t) - Cost_{i^{(j)}}(t)\right), \tag{14}$$

where $U_j(t) = \eta_1 \mathrm{H}\left(C_j - \sum_{k \in \mathcal{K}} \beta_{j,k}(t) c_k\right)$ represents whether the cached services exceed the storage capacity of the agent j, and $Y_{i^{(j)}}(t) = \eta_2 \mathrm{H}\left(\tau_{i^{(j)}}(t) - T_{i^{(j)}}(t)\right)$ represents whether the delay requirement of UE $i^{(j)}$ is satisfied. $\mathrm{H}(\cdot)$ is the Heaviside step function. η_1 and η_2 are the weight coefficients. Consequently, we have $U_j(t) = \eta_1$ and $Y_{i^{(j)}}(t) = \eta_2$ if the constraints of storage capacity and delay requirement are satisfied, respectively.

B. Graph Attention-Based MADDPG Algorithm

In this section, we propose a GAT-MADDPG algorithm that combines the graph attention network (GAT) with multi-agent deep deterministic policy gradient (MADDPG) algorithm to solve the formulated Dec-POMDP problem. The framework of the proposed GAT-MADDPG with centralized training and decentralized execution is illustrated in Fig. 2. Specifically, for each agent j at each time step, given the local observation o_j, the actor network takes action a_j according to the current continuous policy[1]. Then the critic network evaluates the selected actions through a centralized action-value function with extra information about the actions $\boldsymbol{a} = (a_1, a_2, \cdots, a_N)$ and latent feature vectors h_j, and h_j is generated by a GAT layer that

[1]In this paper, the hybrid action space is converted into a homogeneous one through continualization.

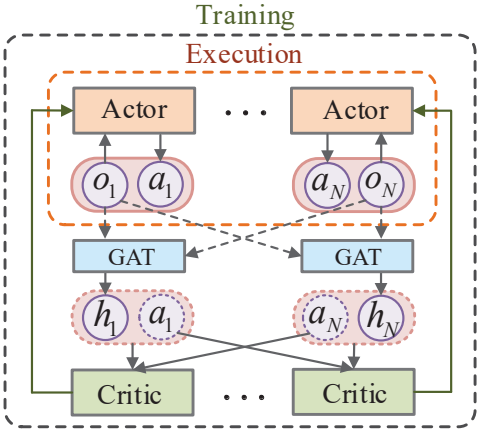

Fig. 2: The framework of the GAT-MADDPG

takes the graph-structured observations $\boldsymbol{o} = (o_1, o_2, \cdots, o_N)$ of all agents as inputs. By stacking the GAT layer, the critic network selectively attends to information of other agents in the environment to generate the Q-value for agent j. The GAT network will be described in detail in the next subsection. Let $\boldsymbol{\pi} = \{\pi_1, \pi_2, \cdots, \pi_N\}$ be the set of continuous policies parameterized by $\boldsymbol{\theta} = \{\theta_1, \theta_2, \cdots, \theta_N\}$. During the training phase, each agent makes action independently based on its local observation to maximize the expected reward, i.e., $\mathcal{J}(\theta_j) = \mathbb{E}[r_j]$. The actor updates the policy $\boldsymbol{\pi}_{\theta_j}$ w.r.t. parameters θ_j (abbreviated as $\boldsymbol{\pi}_j$) by the gradient of the expected reward as

$$\begin{aligned} &\nabla_{\theta_j} \mathcal{J}(\boldsymbol{\pi}_j) \approx \\ &\tfrac{1}{X} \sum_{x=1}^{X} \nabla_{\theta_j} \boldsymbol{\pi}_j\left(o_j^x\right) \nabla_{a_j} Q_j^{\omega_j}\left(h_j^x, \boldsymbol{a}^x\right)\Big|_{a_j = \boldsymbol{\pi}_j\left(o_j^x\right)} \end{aligned} \tag{15}$$

where X represents the number of the mini-batch size that is randomly sampled from the replay buffer, and x indicates the x-th sample. $Q_j^{\omega_j}\left(h_j^x, \boldsymbol{a}^x\right)$ is the centralized action-value function. The replay buffer contains the tuple $(\boldsymbol{o}, \boldsymbol{o}', \boldsymbol{a}, r_1, r_2, \cdots, r_N)$, where $\boldsymbol{o}'$ is the next observation of all agents. Then the GAT and critic networks jointly update parameters ω_j by minimizing the following loss function for each agent j

$$\mathcal{L}(\omega_j) = \frac{1}{X} \sum_{x=1}^{X} \left(Q_j^{\omega_j}\left(h_j^x, \boldsymbol{a}^x\right) - y^x\right)^2, \tag{16}$$

$$y^x = r_j^x + \gamma Q_j^{\omega_j'}\left(h_j'^x, \boldsymbol{a}'^x\right)\Big|_{a_j' = \boldsymbol{\pi}_j'\left(o_j'^x\right)}, \tag{17}$$

where γ is the discount factor, and $\boldsymbol{\pi}'$ is the target policies with parameters $\boldsymbol{\theta}'$. Finally, each agent j updates the target network parameters softly with $\omega_j' = \varsigma \omega_j + (1 - \varsigma)\omega_j'$ and $\theta_j' = \varsigma \theta_j + (1 - \varsigma)\theta_j'$, where ς is update rate.

C. Graph Attention Network

We construct the multi-agent environment as an undirected graph $G(V, E, A)$, where V is the set of nodes and each node represents an agent, E is the set of edges in the graph, and

A is the adjacency matrix. At each time step, the feature of each node j is o_j. In this paper, we use graph attention mechanism [14] to adaptively capture the correlations among agents. Specifically, the attention coefficients between agent j and agent j' can be computed by $e_{j,j'} = att\left(\mathbf{W}o_j, \mathbf{W}o_{j'}\right)$, where $att\left(\cdot\right)$ is the attention mechanism and $\mathbf{W}$ is the corresponding learnable weight matrix. The attention coefficients $e_{j,j'}$ indicate the importance of agent j's features to its neighbor agent j'. Note that we only consider the first-order neighbors of agent j (including agent j). Then the attention coefficients are normalized by using the softmax function to make them easily comparable for different agents as follows

$$\alpha_{j,j'} = \mathrm{softmax}_j\left(e_{j,j'}\right) = \frac{\exp\left(e_{j,j'}\right)}{\sum\limits_{k\in\mathcal{N}} \exp\left(e_{j,k}\right)} \tag{18}$$

To stabilize the learning process in the proposed GAT-MADDPG, the multi-head attention mechanism is applied. Given the normalized attention coefficients, the latent feature vector for agent j with L independent attention mechanisms can be given by

$$h_j = \|_{l=1}^{L}\, \sigma\left(\sum_{j'\in\mathcal{N}} \alpha_{j,j'}^{l} \mathbf{W}^{l}(o_{j'}, a_{j'})\right) \tag{19}$$

where σ is a non-linear function, $\|$ represents concatenation, and l indicates the l-th attention mechanism. By using the graph attention mechanism, the agents assign more attention to valuable information while filtering out the worthless parts [10].

V. SIMULATION RESULTS

A. Simulation Settings

We consider there are 5 BSs in the network, and the coverage radius of each BS is set to 200 m. Each BS serves 5 UEs that are uniformly distributed within its coverage area. The channel gain is set to $d_{i^{(j)},j}^{-3}$, where $d_{i^{(j)},j}$ is the distance between UE $i^{(j)}$ and BS j. We assume that each UE generates one task per time step, and we model the service preference of each UE (i.e., $k_{i^{(j)}}\left(t\right)$) as Zipf distribution [15]. Specifically, denote the popularity rank of service k in BS j as $z_{j,k} \in \mathcal{K}$, then the popularity of the service k is

$$\phi_{j,k} = \frac{z_{j,k}^{-\delta_j}}{\sum\limits_{l\in\mathcal{K}} z_{j,l}^{-\delta_j}}. \tag{20}$$

where δ_j denotes the skewness of popularity in BS j. Then we denote the popularity vector in BS j as $\boldsymbol{\phi}_j = \{\phi_{j,1}, \phi_{j,2}, \cdots, \phi_{j,K}\}$, and each UE generates task with specific service requirement according to $\boldsymbol{\phi}_j$ of its local BS. To model the differentiated task popularity under different BSs, we set different popularity vectors for each BS by setting different service popularity ranks and skewness. Specifically, we assume that BS 1 and BS 2 have the same service popularity rank, and BS 3, BS 4, and BS 5 have the same service popularity rank which is different to BS 1 and BS 2.

TABLE I
SIMULATION PARAMETERS

Parameter	Value	Parameter	Value
K	10	c_k	20 GB
C_j	100 GB	f_j	2.5 GHz
$r_{j,j'}$	2 Mbit/s	$r_{j,0}$	0.5 Mbit/s
w	30 MHz	N_0	-50 dBm
$p_{i(j)}$	$[5, 33]$ dBm	$d_{i(j)}$	$[10, 30]$ Mbit
$\rho_{i(j)}$	700 cycles/bit	$\tau_{i(j)}$	$[10, 30]$ s
w_d	0.05	w_e	0.01
η_1	5	η_2	1
Actor learning rate	0.0001	Critic learning rate	0.001
Replay buffer size	5×10^5	Mini-batch size	256
Discount factor γ	0.9	Update rate ς	0.01

The service popularity rank is randomly sorted by the service catalog in the network. The skewness coefficients are set to $\delta_1 = 0.8$, $\delta_2 = 1$, $\delta_3 = 0.8$, $\delta_4 = 1$, and $\delta_5 = 1.2$, respectively. Besides, the adjacency matrix A only contains elements of 0 and 1, where 1 means that the service popularity rank is the same between agents, otherwise it is 0.

We use the Pytorch platform to implement the proposed GAT-MADDPG algorithm with Adam optimizer. The actor and critic networks are comprised of two fully connected (FC) hidden layers (128 and 64 neurons) with ReLU as the activation function. The output layer of actor network is activated by the tanh activation function. We apply one GAT layer that consists of 2 attention heads. The attention mechanism $att\left(\cdot\right)$ is a single-layer feedforward neural network activated by the Leaky ReLU activation function. The number of episodes is 1000, and the number of time steps in each episode is set to 100. Unless otherwise specified, other parameters in our simulations are listed in Table I.

We compare the performances of the proposed GAT-MADDPG algorithm with the MADDPG algorithm [7]. In MADDPG, each agent is implemented by the DDPG algorithm without GAT network, and the related parameter configurations are the same as GAT-MADDPG for a fair comparison.

B. Results and Analysis

We investigate the performances of GAT-MADDPG from convergence, average cache hit rate, and average cost. The cache hit rate is the percentage of requests served by BSs. Fig. 3 shows the convergence curves of the average reward of GAT-MADDPG and MADDPG as the episode increases. As can be seen from Fig. 3, the proposed GAT-MADDPG curve raises quickly and then stabilized at the 200-th episode, which outperforms the MADDPG algorithm stabilized at the 400-th episode. Besides, the proposed GAT-MADDPG algorithm has higher average reward. The reason is that the GAT-MADDPG can help agents assign more attention to valuable information of other agents to learn better attentive cooperation policy.

The impact of the storage capacity of BSs on the average cache hit ratio is presented in Fig. 4. The curves show an upward trend with the rise in the storage capacity of BSs. The reason is that the large storage capacity allows BSs to cache

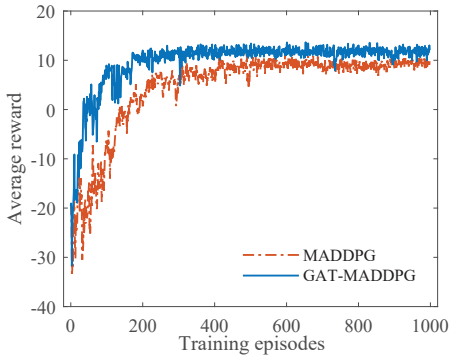

Fig. 3: Training episodes vs. average reward.

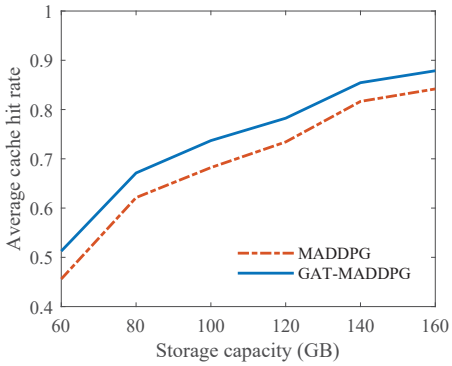

Fig. 4: Storage capacity vs. average cache hit rate.

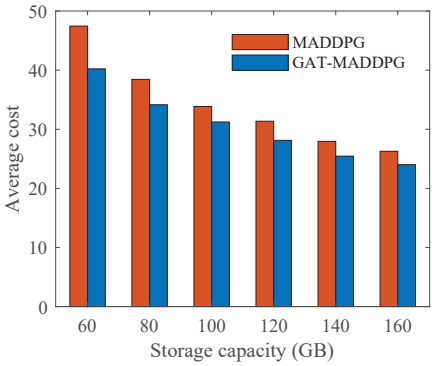

Fig. 5: Storage capacity vs. average cost.

more services to satisfy UEs requests from local or other BSs. Fig. 5 shows the impact of the storage capacity of BSs on the average cost of all UEs in the network. We can notice that the average offloading cost decreases with the increasing storage capacity of BSs. The reason is that UEs can offload task to the local BS with less offloading cost since more services are cached in the local BS. As shown in Fig. 4 and Fig. 5, the GAT-MADDPG based task offloading and service caching algorithm shows superiority over MADDPG algorithm on both metrics average cache hit rate and offloading cost.

VI. CONCLUSIONS

In this paper, we studied the cooperative task offloading and service caching problem for MEC-enabled small cell networks. The optimization problem was formulated to minimize the long-term average cost of computing delay and energy consumption of UEs in the network. We proposed a GAT-MADDPG algorithm to solve this problem. The GAT-MADDPG incorporates a multi-headed graph attention mechanism into the centralized critic network, thereby helping agents selectively assign more attention to valuable information while filtering out the worthless parts. Simulation results demonstrated that the proposed GAT-MADDPG algorithm exhibits an effective performance improvement in metrics such as the average reward, cache hit ratio, and offloading cost. For future work, we will study a more general MEC network that take inter-cell interference into consideration.

ACKNOWLEDGEMENTS

This paper was supported in part by the Program for the National Science Foundation of China (62071077), in part by the Chongqing Research Program of Basic Science and Frontier Technology (cstc2017jcyjBX0005), and in part by Chongqing University of Posts and Telecommunications PhD high-end talent training project (BYJS201806).

REFERENCES

[1] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: the communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322-2358, Aug. 2017.

[2] L. Chen, C. Shen, P. Zhou, and J. Xu, "Collaborative service placement for edge computing in dense small cell networks," *IEEE Transactions on Mobile Computing*, vol. 20, no. 2, pp. 377-390, Feb. 2021.

[3] S. Bi, L. Huang, and Y. A. Zhang, "Joint Optimization of Service Caching Placement and Computation Offloading in Mobile Edge Computing Systems," *IEEE Transactions on Wireless Communications*, vol. 19, no. 7, pp. 4947-4963, July 2020.

[4] G. Zhang, S. Zhang, W. Zhang, Z. Shen, and L. Wang, "Joint Service Caching, Computation Offloading and Resource Allocation in Mobile Edge Computing Systems," *IEEE Transactions on Wireless Communications*, vol. 20, no. 8, pp. 5288-5300, Aug. 2021.

[5] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading Tasks With Dependency and Service Caching in Mobile Edge Computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 11, pp. 2777-2792, Nov. 2021.

[6] D. Lan, A. Taherkordi, F. Eliassen, and L. Liu, "Deep Reinforcement Learning for Computation Offloading and Caching in Fog-Based Vehicular Networks," *IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, Delhi, India, pp. 622-630, 2020.

[7] Q. Mi, N. Yang, H. Zhang, H. Zhang and J. Wang, "Joint Caching and Transmission in the Mobile Edge Network: An Multi-Agent Learning Approach," *IEEE Global Communications Conference (GLOBECOM)*, Madrid, Spain, pp. 1-6, Dec. 2021.

[8] Y. Yan, B. Zhang, C. Li, and C. Su, "Cooperative Caching and Fetching in D2D Communications - A Fully Decentralized Multi-Agent Reinforcement Learning Approach," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 12, pp. 16095-16109, Dec. 2020.

[9] Y. Zhao, R. Li, C. Wang, X. Wang and V. C. M. Leung, "Neighboring-Aware Caching in Heterogeneous Edge Networks by Actor-Attention-Critic Learning," *IEEE International Conference on Communications*, Montreal, QC, Canada, pp. 1-6, Jun., 2021.

[10] H. Chen, Y. Liu, Z. Zhou, D. Hu, and M. Zhang, "GAMA: Graph Attention Multi-agent reinforcement learning algorithm for cooperation," *Applied Intelligence*, vol. 50, no. 12, pp. 4195-4205, 2020.

[11] H. Ryu, H. Shin, and J. Park, "Multi-agent actor-critic with hierarchical graph attention network," *The AAAI Conference on Artificial Intelligence*, New York, USA, vol. 34, no. 05, pp. 7236-7243, Apr., 2020.

[12] S. Xia, Z. Yao, Y. Li and S. Mao, "Online Distributed Offloading and Computing Resource Management With Energy Harvesting for Heterogeneous MEC-Enabled IoT," *IEEE Transactions on Wireless Communications*, vol. 20, no. 10, pp. 6743-6757, Oct. 2021.

[13] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[14] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Lio, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[15] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402-8413, Dec. 2013.