IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received May 30, 2020, accepted June 12, 2020, date of publication June 16, 2020, date of current version June 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3002895

Joint Optimization of Caching and Computation in Multi-Server NOMA-MEC System via Reinforcement Learning

SHILU LI, BAOGANG LI, (Member, IEEE), AND WEI ZHAO, (Member, IEEE)
North China Electric Power University, Baoding 071003, China

Corresponding author: Baogang Li (baogangli@ncepu.edu.cn)

This work was supported in part by the National Natural Science Foundation of China under Grant 61971190, in part by the Fundamental Research Funds for the Central Universities under Grant 2019MS089 and Grant 2017MS159, and in part by the Beijing Natural Science Foundation under Grant 4164101.

ABSTRACT With the development of emerging applications such as augmented reality, more and more computing tasks are sensitive to delay. Caching popular task computation results on the mobile edge computing (MEC) server is an effective solution to meet the latency requirements. When multiple users request the same task, if the computation result is cached on the MEC server, it will return the computation result directly to the user to reduce the delay for repeated computation. In this paper, we use the caching to assist the calculation. Non-orthogonal multiple access (NOMA) is used to further reduce the delay for computation offloading. The optimization problem is formulated as how to make caching and offloading decision to minimize the delay of whole system. In the case of unknown popularity, we use Gated Recurrent Unit (GRU) algorithm to predict the task popularity in time-varying system, and place the computing results of tasks with high popularity on the corresponding server. Based on the predicted popularity, a multi-agent Deep-Q-network (MADQN) algorithm is used to solve the caching and offloading problem. The simulation results show that the prediction error of GRU algorithm can be reduced by increasing the learning rate. Meanwhile, the proposed MADQN can effectively reduce the delay compared with other methods.

INDEX TERMS Mobile edge computing (MEC), non-orthogonal multiple access (NOMA), caching, multi-agent Deep-Q-network (MADQN).

I. INTRODUCTION

With the development of mobile communication technology, more and more new applications have a strict requirement for latency. Mobile edge computing (MEC) is one promising technology to meet the low latency demand. It can reduce the burden on mobile users by offloading computing tasks to the MEC server [1]. Therefore, the research on MEC system has attracted much attention. In [2], the authors considered the problem of multi-user multi-server task offloading and the goal was to minimize the communication delay. The optimization of radio and computational resources in MEC system was studied in [3], which can reduce the total consumed energy. However, one same assumption adopted in [1]–[3] is that the computation tasks for different users are different and the computation results cannot be reused, which may not always hold in practice. In fact, it is very common for users to reuse the computation results with the innovative applications. For example, visitors in augmented reality may request a processed augmented reality output synchronously or asynchronously for better experience; In mobile online games, the game environment can also be requested synchronously by a group of players or asynchronously by individual players. In the meanwhile, considering the limited computing capacity of the MEC server, it is difficult to handle a large number of computing tasks. If these repeated tasks are calculated on the MEC server, it will cause a waste of computing resources.

In order to avoid duplicate transmission and computation of same tasks, caching the computation results of popular tasks is an effective solution to reduce latency. As for the research on caching, most of the papers mainly focus on

The associate editor coordinating the review of this manuscript and approving it for publication was Hongbin Chen.

content caching, but less on the caching of computation results. The content caching, which usually caches videos, does not involve task executing and computation results downloading [4]. The caching of computation results refers to caching popular tasks computation results, which can save the delay of repeated computation [5]. Existing research on the caching of computation results is mainly in MEC systems, which is often used to assist in task offloading and computation [6]. For example, the joint optimization of caching and edge computation in MEC system was studied in [7], [8], where the goal was to reduce total execution delay. The core idea of [9] was that how to design the caching and computation offloading policy to minimize the required average transmission rate. In [10], the authors proposed the joint optimization of caching and computation in a multi-user cache-assisted MEC system, where MEC servers cache computation results for future demands. Edge computing, caching and multicast were considered in [11], which can tackle the wireless bandwidth bottleneck problem. However, the main concern of computation results caching and task offloading on above MEC system is computing and caching capabilities of the edge servers, they do not consider how to use limited spectrum resources to further reduce latency.

The combination of non-orthogonal multiple access (NOMA) and caching has become an effective way to solve the problem of limited spectrum resources. In the caching network, NOMA technology can push more content to the server or users at the same time. Thus, the efficiency of the transmission in wireless caching network can be improved [12]. For caching in NOMA system, previous articles mainly focused on content caching, which takes into account not only the effect of channel conditions, but also the location and content of the caching [12]–[14]. In the recent work [15], the authors proposed the method of deep reinforcement learning to solve power allocation in cache-aided NOMA systems. In terms of NOMA combined with caching, dynamic power control for NOMA transmissions in wireless caching networks was studied in [16], which designed a deep neural network (DNN)-based method to keep a balance between the performance and the computational complexity. However, caching in NOMA systems is generally about content caching. Although it considers the effect of spectrum resources, it ignores the advantage of computation results caching in assisting edge calculation to reduce delay and energy consumption.

Thus, in order to balance the limited spectrum resources and computing resources in the cache network, caching task computation results on NOMA-MEC system is a new trend to reduce the delay and energy consumption. For the research of NOMA-MEC system, most articles mainly focus on offloading computation. In NOMA-MEC system, by offloading the computation tasks to MEC server, energy consumption of users and the delay of processing tasks can be reduced greatly [17]–[20]. The authors of [21] investigated the resource allocation problem in the D2D communication underlaying a NOMA-based MEC system. In [22], the authors designed a scheme to joint optimize computation offloading and time allocation. Up to now, there are very few works investigating the caching of computation results on NOMA-MEC system. The authors of [23] designed a cache-assisted NOMA-MEC framework, where Q-learning and Bayesian learning automata (BLA) based multi-agent Q-learning (MAQ-learning) algorithm was used to address caching, offloading, and resource allocation issues, but it only covers a MEC server and local caching. The cache capacity of a single MEC server is limited, making it difficult to cache much more task computation results. If users request a variety of tasks, it will result in lower hit rate. In this case, the uncached task can only be computed, so the delay cannot be effectively reduced. Therefore, in this paper, we consider the caching of computation results in NOMA-MEC system with multiple MEC servers. Computation results can be shared between servers. If the local server does not cache the computation results of the task requested by the user, it can retrieve the required content from the collaborating server. However, due to the delay cost of data transmission between two servers, there faces the choice whether to retrieve the result from collaborating server, which is huge challenge for the cooperative caching system.

In addition, for caching, we need to consider task popularity, which is an important factor affecting the caching. Task popularity refers to the probability of a task being requested within a specific time [24]. Most articles believe that popularity is known and follows a Zipf distribution [25]–[28]. However, in the actual scenario, due to the dynamics of the task and the mobility of the user, the popularity changes dynamically over time, so it can be considered as a time series [24]. Gated Recurrent Unit (GRU) algorithm is a method of deep learning, which can deal with vanishing gradient and efficiently capture long-term dependencies [29]. Moreover, it has great advantages in time series prediction. Thus, in this paper, we use the GRU algorithm to predict dynamic and unknown popularity. To our best knowledge, GRU algorithm is first used to predict the task popularity. Based on the predicted task popularity, we propose to optimize caching and offloading decisions in order to minimize the delay of whole system. The main contributions of this paper are summarized as follows:

(1) We study the joint caching and offloading problem in a multi-user NOMA-MEC system with multiple MEC servers to minimize the delay of whole system. By extending the local caching of a single server to the cooperative caching of multiple servers, the computation results can be shared between servers.

(2) Considering the fact that the task popularity is dynamic and unknown, we propose a novel GRU algorithm to predict the task popularity. According to the predicted popularity, computing results of tasks with high popularity can be placed on the corresponding server. We prove that by increasing the learning rate, the prediction error can be reduced.

(3) Based on the predicted popularity, we adopt the multi-agent Deep-Q-network (MADQN) algorithm for

caching and offloading decisions. In the multi-agent system, each user in the same base station is treated as an agent. Before selecting the action, agents need to check the case of hitting to determine the optional action. After that, agents can adjust their behavior through the feedback from the environment without knowing other agents' behaviors in advance. Simulation results show that due to the cooperative caching in MADQN, the hit rate is higher compared to only local caching. At the same time, MADQN can effectively reduce the delay compared with other schemes.

The paper is organized as follows. The system model which includes caching model and computation model in Section II. The problem formulation is in Section III. The problem solution is given in Section IV. Section V provides the simulation results. Finally, Section VI concludes the paper.

II. SYSTEM MODEL

We consider a typical caching and offloading model which includes multiple servers as shown in Fig. 1. In this model, there are N users in each base station (BS) and each user is only connected with a single BS. The number of MEC servers is B. We denote the set of users within the same BS and MEC servers by $\mathcal{N} = \{1, 2, \cdots, N\}$, $\mathcal{B} = \{1, 2, \cdots, B\}$, respectively. Each BS is equipped with one MEC server and the MEC servers could be co-located with the cellular BSs [30]. We assume that multiple MEC servers have the same computing and caching capabilities, denoted by C_{MEC} and N_{cache}, respectively. Multiple servers form a cooperative caching area where the cached computation results can be shared. We consider that there is a set of S computing tasks in edge computing system, denoted by $\mathcal{S} = \{1, 2, \cdots, S\}$. Different from [1]–[3], we focus on the scenario where one task may be required by multiple users, hence its computation result can be reusable [31]. Examples of these types of applications have been illustrated in Section I. Assuming that each user only requests the task once and all users request tasks at the same time, while the same task can be requested by different users according to their preferences.

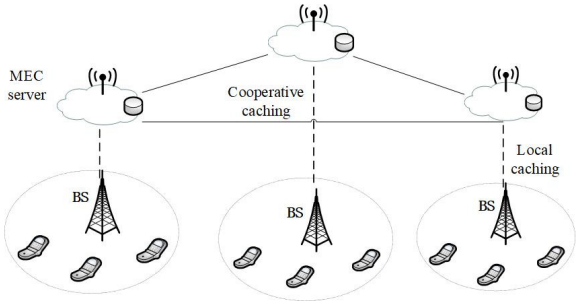

FIGURE 1. An illustration of caching and offloading for mobile edge computing.

Each task $j \in \mathcal{S}$ is characterized by three parameters. Here D_j denotes the size of input data needed for computing task j. W_j denotes the total number of CPU cycles required to accomplish the computation task j. M_j represents the size of computation result for task j. Each user has a computation-intensive and latency-sensitive task to complete. We assume that the user knows the serial number of the requested task. Before the user requests a task, the MEC server has placed the high-popularity computation results. In our model, there are four ways to process computational tasks: local caching, cooperative caching, offloading computation and local computation.

A. CACHING MODEL

In the caching model, there are two caching modes for users to obtain the computation result when choosing the caching decision: the local caching mode and the cooperative caching mode.

1) LOCAL CACHING MODEL

In local caching model, the local MEC server has stored computation results of popular tasks. Before selecting the local caching decision, we need to consider the hit situation. In the case of user i requesting task j, whether the local server has cached task j needs to be checked. We define a variable β_m^j, which indicates whether the local server m has cached task j. $\beta_m^j = 1$ represents local server m has cached task j. Otherwise, $\beta_m^j = 0$. If user i chooses local caching decision when hitting the local caching, the local MEC server will send the calculation result directly to user i. The local caching decisions of N users within the coverage of BS connected to MEC server m is denoted as $Z_m = [z_m^1, z_m^2, \cdots, z_m^N]$, where $z_m^i \in \{0, 1\}$. If $z_m^i = 1$, it means that user i gets computation results from the cache content of the local MEC server m. Otherwise, $z_m^i = 0$. Furthermore, $Z = [Z_1, Z_2, \cdots, Z_B]$ is used to represent local caching decisions on all MEC servers.

Although local caching can cause some delay, the delay of local caching is negligible compared to cooperative caching and offloading computation [32]. This is because compared to cooperative caching, the local MEC server is closer to the user. At the same time, compared to the offloading computation, the download data rate is higher and the size of computation result is much smaller than that of input data. Therefore, we can ignore the delay of local caching.

2) COOPERATIVE CACHING MODEL

In cooperative caching model, the cooperative server has stored computation results of popular tasks. If the user chooses cooperative caching decision when hitting cooperative caching, the local server retrieves the computation results from the cooperating MEC server and returns them back to the user. We define the cooperative caching vectors of N users within the coverage of BS connected to MEC server m as $\alpha_m = [\alpha_{m,n}^1, \alpha_{m,n}^2, \cdots, \alpha_{m,n}^N]$, where $\alpha_{m,n}^i \in \{0, 1\}$. If $\alpha_{m,n}^i = 1$, it represents that user i gets the calculation result of the requested task from the cooperative server n, which collaborates with the local server m. Otherwise, $\alpha_{m,n}^i = 0$. In the system, there are B servers. Therefore, cooperative caching vector on all servers is denoted as $\alpha = [\alpha_1, \alpha_2, \cdots, \alpha_B]$.

When the computation results of the task are cached on the servers, communication between BSs facilitates the sharing of information about the results of caching tasks between MEC servers. For example, the X2 interface between NodeBs in LTE can provide data exchange [24], so it can be used to realize information exchange between servers. In cooperative caching, due to the long distance between servers, the transmission delay cannot be ignored. Therefore, the delay of cooperative caching can be obtained as follows:

$$T^j_{m,n} = \frac{M_j}{r_{m,n}} \tag{1}$$

where $r_{m,n}$ represents download rate between local server m and cooperative server n.

B. COMPUTATION MODEL

Computation results cached on the server can be obtained through task offloading and computation at the MEC server during the historical period. In order to obtain complete computation results, we consider that tasks are either completely offloading or completely local computation. Therefore, there are two ways to process tasks in computing model: local computation and offloading computation.

1) LOCAL COMPUTING MODEL

The N users' local computation vectors within the coverage of BS connected to MEC server m can be defined as $Y_m = [y^1_m, y^2_m, \cdots, y^N_m]$, where $y^i_m \in \{0, 1\}$, $y^i_m = 1$ means that user i within the coverage of BS connected to MEC server m decides to execute the task by itself. Otherwise, $y^i_m = 0$. Therefore, the local computation vector on all servers can be represented as $Y = [Y_1, Y_2, \cdots, Y_B]$.

If user i chooses to execute its task j locally, the local execution delay of task j is

$$T^l_{i,j} = \frac{W_j}{f^l_i} \tag{2}$$

where f^l_i represents the computation capacity (i.e. CPU cycles per second) of user i in the local computing phase.

We have $E^l_{i,j}$ as the corresponding energy consumption of task j, which is expressed as

$$E^l_{i,j} = e(f^l_i)^2 W_j \tag{3}$$

According to work [33], the energy consumption per computing cycle is $\varepsilon = e(f^l_i)^2$, where e is the energy coefficient, which depends on the chip architecture. We set $e = 10^{-27}$ [33].

2) OFFLOADING COMPUTING MODEL

We define the task offloading strategy on server m as $X_m = [x^1_m, x^2_m, \cdots, x^N_m]$, where $x^i_m \in \{0, 1\}$, $x^i_m = 1$ indicates that the task requested by the user i is offloaded to the local server m. Otherwise, $x^i_m = 0$. Then, we can obtain task offloading decisions on all servers, which can be expressed as $X = [X_1, X_2, \cdots, X_B]$. Assuming that there are N^m_{up} users to offload the computation tasks to the local server m. It can be represented as $\mathcal{N}^m_{up} = \left\{1, 2, \cdots, N^m_{up}\right\}$, where $N^m_{up} = \sum_{i=1}^{N} x^i_m$.

If user i chooses to execute task j by offloading computing, the whole offloading approach will be divided into three steps. Firstly, user i needs to offload input data to local BS. In order to improve the spectrum utilization and further reduce the transmission delay, we adopt the NOMA technique to transmit information to local BS. In the NOMA model, multiple users transmit simultaneously and share the same uplink channel. At the BS, in order to distinguish the superimposed signals, the successive interference cancellation (SIC) is adopted for decoding, that is, the user with better channel quality is decoded firstly, and it is subtracted from the received signal so that it will not interfere with the user with worse channel quality. Then, in this way, the signals of users with worse channel quality are decoded. For data offloading, suppose that channel gain between user i and MEC server m is $\left|h^i_m\right|^2$, $i \in \mathcal{N}^m_{up}$. Without loss of generality and for the ease of discussion, there is $\left|h^1_m\right|^2 \geq \left|h^2_m\right|^2 \geq \cdots \geq \left|h^{N^m_{up}}_m\right|^2$. The achievable transmission rate between the mobile user i and MEC server m, denoted as R^i_m, can be written as:

$$R^i_m = Blog_2(1 + \frac{p_i \left|h^i_m\right|^2}{\sum_{g=i+1}^{N^m_{up}} p_g \left|h^g_m\right|^2 + \sigma^2}) \tag{4}$$

where p_i represents the transmit power of user i, and σ^2 denotes the power of additive noise. The transmission delay user i offloads the task j to the local BS is

$$T^o_{i,j} = \frac{D_j}{R^i_m} \tag{5}$$

The corresponding transmission energy consumption is

$$E^o_{i,j} = p_i T^o_{i,j} \tag{6}$$

Then the MEC server allocates part of computational resource to execute the computing task. The required time is the processing delay of the MEC server, which can be given as

$$T^{mec}_{i,j} = \frac{W_j}{f^{mec}_{m,i}} \tag{7}$$

where $f^{mec}_{m,i}$ is defined as the computational resource allocated to user i by the MEC server m.

Finally, the MEC server returns the computation result to user i. This step is the same as the local caching process, so the delay of this step is also neglected.

III. PROBLEM FORMULATION

According to above discussion, considering caching and computing, the delay of user i within the coverage of BS connected to MEC server m is

$$T^i_m = (1 - z^i_m)[\alpha^i_{m,n} T^j_{m,n} + x^i_m (T^o_{i,j} + T^{mec}_{i,j}) + y^i_m T^l_{i,j}] \tag{8}$$

The delay of server m is the maximum delay of N users, so the delay of server m can be expressed as

$$T_m = max\left\{T_m^1, T_m^2, \cdots, T_m^N\right\} \tag{9}$$

In this way, the delay of whole system is the maximum delay of all servers, which is expressed as follows:

$$T = max\left\{T_1, T_2, \cdots, T_B\right\} \tag{10}$$

Our goal is to minimize the delay of whole system under the constraints of computation resource, cache capacity, and users' energy consumption. Thus, the problem can be formulated as follows:

$$\min_{X,Y,Z,\alpha} \quad T \tag{11}$$

$$s.t.\ x_m^i, y_m^i, z_m^i \in \{0,1\}, \quad \forall i \in \mathcal{N}, m \in \mathcal{B} \tag{12}$$

$$\alpha_{m,n}^i \in \{0,1\}, \quad \forall i \in \mathcal{N}, m, n \in \mathcal{B}, m \neq n \tag{13}$$

$$x_m^i + y_m^i + z_m^i + \alpha_{m,n}^i = 1 \tag{14}$$

$$\sum_{i=1}^{N_{up}^m} f_{m,i}^{mec} \leq C_{MEC}, \quad \forall m \in \mathcal{B} \tag{15}$$

$$\sum_{j=1}^{S} \beta_m^j M_j \leq N_{cache}, \quad \forall m \in \mathcal{B} \tag{16}$$

$$\sum_{i=1}^{N} (x_m^i E_{i,j}^o + y_m^i E_{i,j}^l) \leq E_{max}, \quad \forall m \in \mathcal{B} \tag{17}$$

In this paper, we mainly use the caching to assist the calculation, without special considering the effect of the caching policy. Therefore, objective function is set to one-shot latency. In the above formulation, (12) and (13) denote that offloading, local computation, local caching, and cooperative caching decisions are binary variables, respectively. According to (14), the user only chooses one of the decisions to get the computation result of requested task. (15) makes sure that the sum of computational resource allocated to the offloading users can not exceed the computing capacity of MEC server. (16) guarantees that the cached computation results do not exceed the caching capacity of the MEC server. In formula (17), it represents the energy consumption of N users in the same BS is limited to maximum energy consumption.

Since the optimization problem consists of 0-1 variables, thus, problem (11) is non-convex and NP-hard. It is difficult to solve this problem by using traditional optimization methods, while reinforcement learning method has a good advantage in dealing with this problem. Therefore, the algorithm based on reinforcement learning is used to find the optimal policy.

IV. PROBLEM SOLUTION

In this section, we firstly use the GRU algorithm to predict the task popularity, and determine the computation results cached on MEC servers according to the popularity. Then, based on the cached computation results on the servers, MADQN method is used to find the optimal caching and offloading decision.

A. GATED RECURRENT UNIT

Before users request these tasks, we must decide what to cache to the servers without knowing the task popularity. In order to increase caching hit rate and reduce system delay, we need to predict the task popularity. Popularity is changing over time in the actual scenario. Therefore, in this model, we can consider task popularity as a time series. GRU algorithm is widely used in the prediction of time series, so it can be used to predict the task popularity. GRU is a kind of Recurrent Neural Network (RNN), which can solve such problems as the gradient in long-term memory and back propagation. GRU model is shown in Fig 2.

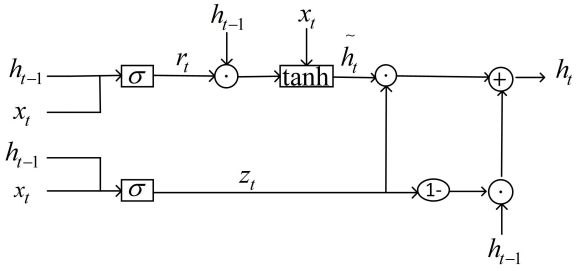

FIGURE 2. Detail structure of GRU cell.

A GRU algorithm contains two gates: a reset gate r_t and an update gate z_t.

$$r_t = \sigma(W_r \cdot [h_{t-1}, x_t]) \tag{18}$$

$$z_t = \sigma(W_z \cdot [h_{t-1}, x_t]) \tag{19}$$

where W_r and W_z represent the weights, while h_{t-1} and x_t denote the input of network, $\sigma(\cdot)$ is a logistic sigmoid function.

The reset gate determines how the new input is combined with the previous memory, and the update gate determines how much previous memory counts. The hidden state can be expressed as

$$h_t = (1 - z_t) \odot h_{t-1} + z_t \odot \tilde{h}_t \tag{20}$$

$$\tilde{h}_t = tanh(W \cdot [r_t \odot h_{t-1}, x_t]) \tag{21}$$

where $tanh(\cdot)$ denotes the hyperbolic tangent function, $\odot$ refers to element-wise multiplication.

In the system, the task popularity before users request these tasks is unknown, and we collect the tasks requested by users at historical time and the computation results of different tasks. In GRU algorithm, it uses historical popularity as inputs and outputs is the task popularity in the near future. $\left\{S_j(1), S_j(2), \cdots, S_j(T_M)\right\}$ is an input sequence of length T_M. The output sequence is expressed as $\left\{S_j(T_M+1), S_j(T_M+2), \cdots, S_j(T_M+T_N)\right\}$, which output sequence of length is T_N, where $S_j(t)$ indicates the popularity of the task j during the time t.The cross entropy error is used as a loss function [34]:

$$L = -\sum_{j=1}^{N_t} p(j) log q(j) \tag{22}$$

where the popularity of the predicted task j is expressed in terms of $q(j)$, $p(j)$ is the popularity of the actual task j. The cross entropy is used to compare the popularity of the current training with the actual distribution of the popularity when the future moment arrives. By minimizing the loss function, the predicted value is closer to the real value and the prediction is more accurate. During the training, we take the derivative of loss function through back-propagation with respect to all parameters, and update parameters with stochastic gradient descent.

According to the above method, GRU algorithm is used to predict the popularity of different tasks in the near future periods. In the case of not exceeding the cache capacity of the MEC server, that is, in order to satisfy the constraint of formula (16), it is important to determine what computation results are stored on the server. Considering the cache capacity of the server and the size of computation results, the computing results of the task are cached on the server in order of popularity from large to small, so as to cache the computing results of the task as much as possible. The above method is usually suitable for storing separately popular content from its own service area at each MEC server. For multiple servers, when the number of servers is small, for example, three servers, these servers can collaborate to cache popular computation results. For the same computing task at each MEC server, we consider its popularity and cache computation results with high popularity on the server where it is located. If the popularity is same on different servers, the computation results is randomly cached on one of the servers. In addition, we also need to consider the location of multiple servers when caching popular content, because the distance between servers can affect the transmission delay. However, for scenarios with a large number of servers in the model, it is very challenging for multiple servers to cache content through mutual cooperation, which is one of our future research directions. Before all users request tasks, these servers have placed the popular computation results, which is beneficial to hit the local caching and cooperative caching.

B. MULTI-AGENT DEEP REINFORCEMENT LEARNING

When the system contains a lot of users, our goal is to find a strategy to minimize the delay in the whole system. In different BSs, the tasks are handled in the same way, such as local caching, cooperative caching, local computation or offloading computation. Therefore, we take the process of handling tasks by users in a BS as an example, and the method of multi-agent reinforcement learning (MARL) is used to optimize the caching and offloading decisions. In MARL, we treat each user in the same BS as an agent. In practice, when users are agents of deep reinforcement learning, in order to reduce high running delay and energy consumption, there are usually lightweight methods combined with deep reinforcement learning to minimize the objective function, which is beyond the scope of the paper.

Considering each user in the same BS as an agent and multiple users in the same BS, the network can be considered as a multi-agent system. In the multi-agent system, the ith agent is represented as a tuple $< s_i, a_i, r_i, \chi_i >$, where s_i is the set of state, a_i is a finite set of action, r_i represents the reward, and χ_i refers to the state transition function of agent i, which describes the probability that the system will reach the next state after taking action a_i in the current state s_i. As shown in Fig. 3, each agent can get state information s_i^t from the environment. After that, it chooses its own action a_i^t. In a multi-agent system, multiple users' decisions form a set of joint actions $A_t = \{a_1^t, a_2^t, \cdots, a_N^t\}$. Based on the current state and action, each agent earns the reward r_i^t, and the system gets to the next state with the transfer probability χ_i. The state transitions of their common environment depend on the agents' joint actions.

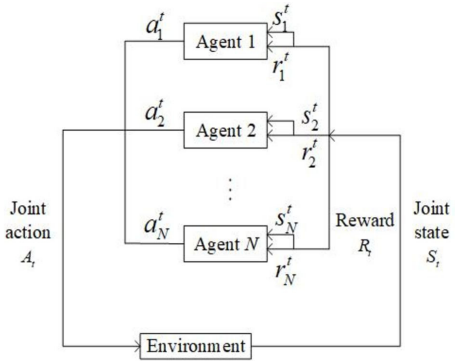

FIGURE 3. The overview of MARL for caching and computation scheme.

Combined with the model, the corresponding relationship with state, action and reward is as follows:

State: For agent i, $s_i(t) = (\mu(t), T_m^i(t), \varphi(t), \zeta(t))$ represents the state. We denote $\mu_m(t)$ as the serial number of tasks stored on the server m at time step t. Therefore, serial number of tasks stored on all servers at time step t is expressed as $\mu(t) = \{\mu_1(t), \mu_2(t), \cdots, \mu_B(t)\}$. In this paper, we have cached corresponding computation results on each server in advance by using the GRU algorithm. Cached computation results can be shared between multiple servers. Through the communication between BSs, the BS can obtain the serial number of cached tasks on other servers. $T_m^i(t)$ represents the delay of user i on the local server m. $\varphi(t)$ is the available computational capacity of the MEC server, which can be represented as $\varphi(t) = C_{MEC} - \sum_{i=1}^{N_{up}^m} f_{m,i}^{mec}(t)$. $\zeta(t)$ denotes available energy consumption of N users in the same BS, and can be computed as $\zeta(t) = E_{max} - \sum_{i=1}^{N}[x_m^i(t)E_{i,j}^o(t) + y_m^i(t)E_{i,j}^l(t)]$.

Action: At time step t, after observing the state of agent i, it can select one action. For agent i, there are several alternative actions: $a_i(t) = (x_m^i(t), y_m^i(t), z_m^i(t), \alpha_{m,n}^i(t))$, where $x_m^i(t)$ is task offloading decision, $y_m^i(t)$ represents local computation decision. As for $z_m^i(t)$ and $\alpha_{m,n}^i(t)$, they represent local caching and cooperative caching decisions, respectively.

However, not all of these actions are optional for all agents. Offloading and local computation are actions that each agent can choose. For caching decisions, we need to judge the case of the hit. Depending on the state information and the task requested by the user, the agent can know whether the local caching or cooperative caching is hit. Only when the cache is hit can the agent select the cache decision. For example, the agent i only hits the local caching. In this way, the cooperative cache is not selected when the action selection is made. Its optional actions are local caching, local computation, and offloading. If the cooperative caching is selected, it cannot get the computation results on the cooperative server. At the same time, the environment cannot feedback the reward information to agent i. Thus, it will cause running error.

When choosing the action, we consider that computing capacity of MEC server and energy consumption of N users in the same BS are limited, once other agents perform local computation or offloading computation, they can take up computing resources and energy consumption, leading to a possible change in the strategy for agent i. In other words, the behavior of other agents will affect the decision of agent i. In MARL, agent i does not know the actions and rewards of the other $N-1$ agents. It can only use its own state information when learning a policy. In order not to exceed the computational capacity of MEC server and the energy consumption of N users in the same BS, we use the available computational capacity of the MEC server $\varphi(t)$ and the rest of energy consumption of N users $\zeta(t)$ as state information. According to the state information, agent i makes its own action choices. Based on the reward of the feedback from the environment, the selected action is constantly adjusted in order to minimize the delay.

Reward: Agent i makes its own decisions based on observations of the environment. Then, agent i receives a reward to evaluate the selected action. In reinforcement learning, our goal is to maximize rewards. In the system, we want to minimize the objective function. To meet the requirements of both, we set the reward of agent i as:

$$r_i(t) = T_i^{local}(t) - T_i(t) \tag{23}$$

where $T_i^{local}(t)$ is the latency when the task of user i is executed locally, $T_i(t)$ represents the actual delay of user i at time step t.

In this paper, there are a large number of users in the BS, and each agent makes its own action choices based on state information. Therefore, the state and action spaces are huge, causing the problem of dimensional disaster. In this case, due to the large number of states and actions, it is impractical for multi-agent Q-learning to find the optimal strategy by looking up tables. The multi-agent Deep-Q-network algorithm can just solve the above problem.

In the MADQN based caching and task offloading scheme, the output of the convolutional neural networks (CNN) is used for estimating the optimal Q-values. When finding the policy, we apply the $\epsilon-greedy$ algorithm. The agent chooses the optimal caching and task offloading policy with high probability $1-\epsilon$. Otherwise, it selects a random policy with probability ϵ. In the training phase, the experience of agent i is $e_i = (s_i, a_i, r_i, s_i')$, where s_i, a_i, r_i correspond to the state, action, and reward of agent i. s_i' represents the next state of agent i. The experience of all agents is stored in the experience replay memory $D = \{e_1, e_2, \cdots, e_n\}$, where n is the number of agents. During training, small batches of experience are randomly selected from D. To minimize the loss function, for the update of parameter θ, it is obtained according to the stochastic gradient descent method based on the previous caching and offloading experience in the memory pool. In the MADQN network, the parameters of the current Q-network are updated in real time. Every M steps, the parameters of the current Q-network are copied to the target Q-network. Stop training once the parameter θ converges. The details of our proposed MADQN for caching and offloading decision is provided in Algorithm 1. According to this method, caching and offloading are performed on users in other BSs. Since the delay of whole system is composed of the delay of multiple BSs, the purpose of minimizing the delay of whole system can be achieved by reducing the delay of different BSs.

Algorithm 1 MADQN for Caching and Offloading Policy

```
Initialization:
for all i ∈ N do
    Initialize state s_i, action a_i, reward r_i
    Initialize replay memory D, current Q-network Q(s, a; θ), target Q-network Q̂(s', a'; θ̂)
end for
while θ is not convergence do
    for iteration do
        for each agent i = 1 : N do
            Observe initial state s_i
            if with probability 1 − ϵ then
                a_i = max_a Q(s, a; θ)
            else
                Select random action a_i
            end if
            Update the reward r_i according to (23)
            Observe the next state s_i'
            Store transition (s_i, a_i, r_i, s_i') in D
            Sample a min-batch of M transitions from D
            Using stochastic gradient to minimize the loss
            L = (r_i + γ max_a' Q̂_i(s', a'; θ̂) − Q_i(s, a; θ))^2
            Update the state s_i ← s_i'
        end for
    end for
end while
```

V. PERFORMANCE EVALUATION

A. EXPERIMENT SETUP

In this section, we present simulation results to evaluate the performance of the NOMA-MEC system for caching

and computation. In the simulation, we set the number of MEC servers to 3 and the number of users served by the same BS to 10. Multiple BSs are randomly distributed in a 800 × 800 m^2 area. The coverage of each BS is a circular area with radius of 100m, and the BS connected with a server is located in the center of this circular area. Coverage areas between different BSs do not overlap. The positions of users in the same BS are randomly distributed in a circular area.

The computing capacity of each MEC servers is set to 5GHz/sec. The bandwidth $B = 10MHz$ and the corresponding noise power $\sigma^2 = 2 \times 10^{-13}$. The size of input task D_j (kbits) follows the uniform distribution between (300,500). In addition, the total number of CPU cycles W_j (in Megacycles) obeys the uniform distribution between (1000, 1300). In the local computing phase, the CPU frequency of each user $f_i^l = 0.6GHz/sec$. The user's transmission power during offloading is set to 0.5W.

B. PREDICTION OF TASK POPULARITY

In the NOMA-MEC system, we consider the location of users to be fixed, but the task requested by users changes constantly over time. We use the GRU algorithm to predict the task popularity. For the dataset, through the analysis of data characteristics, we use software to simulate the communication scene. In the simulated environment, multiple users and a BS are added. The user's channel is set to Rayleigh fading. We collect the number of times each task is requested at different times and put them into a dataset. Due to the huge difference of these data, therefore, the preprocessing work is normalizing them into [0, 1]. Next, we take the example of predicting the popularity of task j on MEC server m to prove that the GRU algorithm has a good prediction effect.

In the GRU algorithm, we use the Adam optimization method and set the learning rate as 0.001, 0.005, and 0.009, respectively. Based on the popularity of historical period, the popularity in near future period is predicted by GRU algorithm. The predicted popularity with different learning rates is shown in Fig. 4(a). In Fig. 4(b), it compares the loss function with different learning rate. The loss function reflects the difference between the true value and the predicted value. According to Fig. 4(b), we can see that the loss function of GRU algorithm with learning rate of 0.009 is smaller and the prediction is more accurate. Therefore, a better prediction effect can be achieved by appropriately increasing the learning rate within a certain range.

C. CACHING AND OFFLOADING DECISION EVALUATION

By predicting the popularity, we place the computation results of the task on the corresponding server. We consider two kinds of caching decisions: cooperative caching and only local caching.

Cooperative caching: Cached results can be shared between multiple servers. During the caching process, the user can get the computation results not only from the local server, but also from cooperative server.

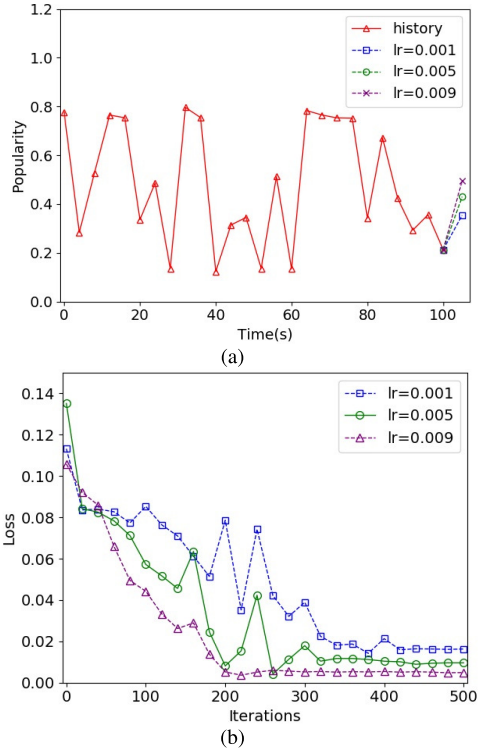

FIGURE 4. Prediction effect of GRU. (a)Task popularity prediction over different learning rate. (b) Loss function of GRU with different learning rate.

Only local caching: No other MEC servers cooperate with the local server. The user can only obtain calculation result of the requested task from local server during the caching process.

From Fig. 5, we can see that the hit rate increases as the cache capacity of the MEC server increases. In addition, the hit rate of cooperative caching and only local caching is approximately 80% and 45% respectively when the cache capacity of each MEC server is up to 25Mbits. The hit rate of the cooperative caching is higher than that of the only local caching. Besides, the difference becomes more significant with the cache capacity increases. Therefore, we can come to the conclusion that cooperative caching can greatly improve the hit rate, especially for large cache capacity of MEC server.

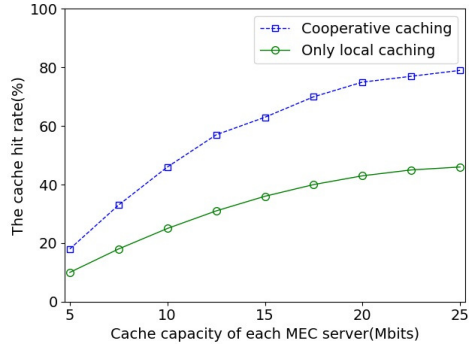

FIGURE 5. Cache hit rate over different cache capacity of MEC server

In Fig.6, we compare the performance of MADQN with the following two schemes. 1) Multi-agent Q-learning (MAQ), which finds the optimal policy by looking at the Q-table in order to maximize the cumulative rewards. 2)Greedy algorithm (GA), which selects the optimal decisions in the current state, without considering whether it is the best decision for the whole system. From the Fig.6, we can see that the delay of GA, MAQ and MADQN algorithm are approximately 0.75s, 0.65s and 0.55s respectively through successive iterations. Therefore, MADQN scheme outperforms MAQ and GA schemes in terms of the delay. This is because compared with MAQ algorithm, MADQN uses the output of CNN to estimate Q-values, instead of calculating Q-values by looking up Q-table, which avoids the situation that it is difficult to find the optimal decision when the state-action space is large. Meanwhile, compared with the GA algorithm, MADQN can get the optimal decision for the whole system through successive iterations, while GA algorithm can only ensure to obtain the optimal policy in the current state. Besides, it can also be observed that MADQN algorithm is superior to GA and MAQ algorithms in convergence speed.

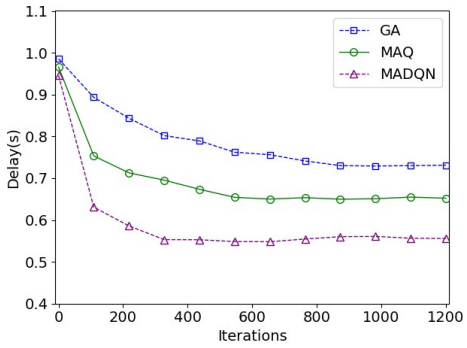

FIGURE 6. The relationship between iterations and the delay of whole system.

In the following, we compare the delay for varying input size from 1Mbits to 5Mbits in Fig. 7. We can observe that the delay of MADQN is lower than that of GA, MAQ and only computation. Therefore, the proposed MADQN algorithm is better than other schemes. As the number of input tasks increases, the delay increases gradually. As a result, the goal of reducing the delay can be achieved by appropriately reducing the size of input tasks.

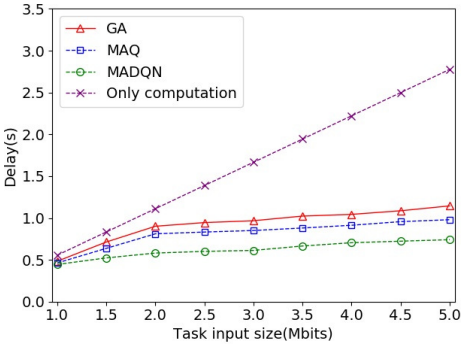

FIGURE 7. The delay of whole system over different task input size.

VI. CONCLUSION

In this article, we address the issue of minimizing delay through the cooperation of multiple MEC servers in NOMA-MEC system. Considering that the popularity is unknown, we use the GRU algorithm to predict the popularity of tasks, and cache the computing results of tasks with high popularity to the corresponding servers. Then, based on the tasks' computation results cached by the server, we propose a multi-agent reinforcement learning method. In order to solve the problem that traditional MAQ algorithm is difficult to deal with large state-action space, we adopted MADQN algorithm, which can effectively reduce the delay. Simulation results show that the prediction of GRU algorithm is more accurate by increasing the learning rate. Meanwhile, compared with only local caching, cooperative caching can effectively improve hit rate. Moreover, MADQN algorithm shows a good advantage in reducing delay and convergence in comparison with other schemes.

REFERENCES

[1] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 771–786, Apr. 2019.
[2] K. Li, M. Tao, and Z. Chen, "Exploiting computation replication for mobile edge computing: A fundamental computation-communication tradeoff study," 2019, *arXiv:1903.10837*. [Online]. Available: http://arxiv.org/abs/1903.10837
[3] O. Munoz, A. Pascual-Iserte, and J. Vidal, "Optimization of radio and computational resources for energy efficiency in latency-constrained application offloading," *IEEE Trans. Veh. Technol.*, vol. 64, no. 10, pp. 4738–4755, Oct. 2015.
[4] T. X. Tran and D. Pompili, "Adaptive bitrate video caching and processing in mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 18, no. 9, pp. 1965–1978, Sep. 2019.
[5] Y. Lan, X. Wang, D. Wang, Z. Liu, and Y. Zhang, "Task caching, offloading, and resource allocation in D2D-aided fog computing networks," *IEEE Access*, vol. 7, pp. 104876–104891, 2019.
[6] Y. Liao, X. Qiao, L. Shou, X. Zhai, Q. Ai, Q. Yu, and Q. Liu, "Caching-aided task offloading scheme for wireless body area networks with MEC," in *Proc. NASA/ESA Conf. Adapt. Hardw. Syst. (AHS)*, Jul. 2019, pp. 49–54.
[7] H. Zhao, Y. Wang, and R. Sun, "Task proactive caching based computation offloading and resource allocation in mobile-edge computing systems," in *Proc. 14th Int. Wireless Commun. Mobile Comput. Conf. (IWCMC)*, Jun. 2018, pp. 232–237.
[8] W. Chen and L. Han, "Time-efficient task caching strategy for multi-server mobile edge cloud computing," in *Proc. IEEE 21st Int. Conf. High Perform. Comput. Communications; IEEE 17th Int. Conf. Smart City, IEEE 5th Int. Conf. Data Sci. Syst. (HPCC/SmartCity/DSS)*, Aug. 2019, pp. 1429–1436.
[9] Y. Sun, Z. Chen, M. Tao, and H. Liu, "Communication, computing and caching for mobile VR delivery: Modeling and trade-off," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.
[10] Y. Cui, W. He, C. Ni, C. Guo, and Z. Liu, "Energy-efficient resource allocation for cache-assisted mobile edge computing," in *Proc. IEEE 42nd Conf. Local Comput. Netw. (LCN)*, Oct. 2017, pp. 640–648.
[11] Y. Sun, Z. Chen, M. Tao, and H. Liu, "Bandwidth gain from mobile edge computing and caching in wireless multicast systems," 2019, *arXiv:1901.09738*. [Online]. Available: http://arxiv.org/abs/1901.09738
[12] Z. Zhao, M. Xu, W. Xie, Z. Ding, and G. K. Karagiannidis, "Coverage performance of NOMA in wireless caching networks," *IEEE Commun. Lett.*, vol. 22, no. 7, pp. 1458–1461, Jul. 2018.
[13] L. Xiang, D. W. K. Ng, X. Ge, Z. Ding, V. W. S. Wong, and R. Schober, "Cache-aided non-orthogonal multiple access: The two-user case," *IEEE J. Sel. Topics Signal Process.*, vol. 13, no. 3, pp. 436–451, Jun. 2019.

[14] Z. Ding, P. Fan, G. K. Karagiannidis, R. Schober, and H. V. Poor, "NOMA assisted wireless caching: Strategies and performance analysis," *IEEE Trans. Commun.*, vol. 66, no. 10, pp. 4854–4876, Oct. 2018.
[15] K. N. Doan, M. Vaezi, W. Shin, H. V. Poor, H. Shin, and T. Q. S. Quek, "Power allocation in cache-aided NOMA systems: Optimization and deep reinforcement learning approaches," *IEEE Trans. Commun.*, vol. 68, no. 1, pp. 630–644, Jan. 2020.
[16] Y. Fu, W. Wen, Z. Zhao, T. Q. S. Quek, S. Jin, and F.-C. Zheng, "Dynamic power control for NOMA transmissions in wireless caching networks," *IEEE Wireless Commun. Lett.*, vol. 8, no. 5, pp. 1485–1488, Oct. 2019.
[17] Z. Ding, J. Xu, O. A. Dobre, and H. V. Poor, "Joint power and time allocation for NOMA–MEC offloading," *IEEE Trans. Veh. Technol.*, vol. 68, no. 6, pp. 6207–6211, Jun. 2019.
[18] P. Yang, L. Li, W. Liang, H. Zhang, and Z. Ding, "Latency optimization for multi-user NOMA-MEC offloading using reinforcement learning," in *Proc. 28th Wireless Opt. Commun. Conf. (WOCC)*, May 2019, pp. 1–5.
[19] F. Wang, J. Xu, and Z. Ding, "Multi-antenna NOMA for computation offloading in multiuser mobile edge computing systems," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 2450–2463, Mar. 2019.
[20] A. Kiani and N. Ansari, "Edge computing aware NOMA for 5G networks," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1299–1306, Apr. 2018.
[21] X. Diao, J. Zheng, Y. Wu, and Y. Cai, "Joint computing resource, power, and channel allocations for D2D-assisted and NOMA-based mobile edge computing," *IEEE Access*, vol. 7, pp. 9243–9257, 2019.
[22] Y. Wu, K. Ni, C. Zhang, L. P. Qian, and D. H. K. Tsang, "NOMA-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation," *IEEE Trans. Veh. Technol.*, vol. 67, no. 12, pp. 12244–12258, Dec. 2018.
[23] Z. Yang, Y. Liu, Y. Chen, and N. Al-Dhahir, "Cache-aided NOMA mobile edge computing: A reinforcement learning approach," 2019, *arXiv:1906.08812*. [Online]. Available: http://arxiv.org/abs/1906.08812
[24] W. Jiang, G. Feng, S. Qin, and Y. Liu, "Multi-agent reinforcement learning based cooperative content caching for mobile edge networks," *IEEE Access*, vol. 7, pp. 61856–61867, 2019.
[25] M. Gregori, J. Gomez-Vilardebo, J. Matamoros, and D. Gunduz, "Wireless content caching for small cell and D2D networks," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 5, pp. 1222–1234, May 2016.
[26] R. Amer, M. M. Butt, M. Bennis, and N. Marchetti, "Inter-cluster cooperation for wireless D2D caching networks," *IEEE Trans. Wireless Commun.*, vol. 17, no. 9, pp. 6108–6121, Sep. 2018.
[27] W. Jiang, G. Feng, and S. Qin, "Optimal cooperative content caching and delivery policy for heterogeneous cellular networks," *IEEE Trans. Mobile Comput.*, vol. 16, no. 5, pp. 1382–1393, May 2017.
[28] A. Ndikumana, S. Ullah, T. LeAnh, N. H. Tran, and C. S. Hong, "Collaborative cache allocation and computation offloading in mobile edge computing," in *Proc. 19th Asia–Pacific Netw. Oper. Manage. Symp. (APNOMS)*, Sep. 2017, pp. 366–369.
[29] E. Pan, Y. Ma, X. Dai, F. Fan, J. Huang, X. Mei, and J. Ma, "GRU with spatial prior for hyperspectral image classification," in *Proc. IEEE Int. Geosci. Remote Sens. Symp. (IGARSS)*, Jul. 2019, pp. 967–970.
[30] T. Hou, G. Feng, S. Qin, and W. Jiang, "Proactive content caching by exploiting transfer learning for mobile edge computing," *Int. J. Commun. Syst.*, vol. 31, no. 11, pp. 3706–3712, 2018.
[31] Y. Hao, M. Chen, L. Hu, M. S. Hossain, and A. Ghoneim, "Energy efficient task caching and offloading for mobile edge computing," *IEEE Access*, vol. 6, pp. 11365–11373, 2018.
[32] J. Xu, L. Chen, and S. Ren, "Online learning for offloading and autoscaling in energy harvesting mobile edge computing," *IEEE Trans. Cognit. Commun. Netw.*, vol. 3, no. 3, pp. 361–373, Sep. 2017.
[33] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.
[34] C. Zhang, H. Pang, J. Liu, S. Tang, R. Zhang, D. Wang, and L. Sun, "Toward edge-assisted video content intelligent caching with long short-term memory learning," *IEEE Access*, vol. 7, pp. 152832–152846, 2019.

SHILU LI received the B.Eng. degree in electronics and communication engineering from Hebei University, Baoding, China, in 2018, where she is currently pursuing the M.S. degree in communication and information engineering. Her research interests include deep reinforcement learning and wireless communication.

BAOGANG LI received the B.Eng. degree from North China Electric Power University, China, in 2006, and the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2012. From 2016 to 2017, he visited the Centre for IoT and Telecommunications, University of Sydney, as a Visiting Scholar. He is currently an Associate Professor with the School of Electrical and Electronic Engineering, North China Electric Power University. His research interests include wireless communication, the industry IoT, and smart grid communication.

WEI ZHAO received the B.Eng. degree from North China Electric Power University, China, in 2008, and the Ph.D. degree from the Beijing University of Posts and Telecommunications, Beijing, China, in 2011. From 2016 to 2017, he visited King's College London, U.K., as a Visiting Scholar. In 2011, he joined the School of Electrical and Electronic Engineering, North China Electric Power University. His research interests include wireless security, massive MIMO, NOMA, and energy efficiency of wireless communications.