IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

SPECIAL SECTION ON ADVANCES ON HIGH PERFORMANCE WIRELESS NETWORKS FOR AUTOMATION AND IIOT

Received 9 May 2023, accepted 4 June 2023, date of publication 9 June 2023, date of current version 14 June 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3284461

RESEARCH ARTICLE

A3C-Based and Dependency-Aware Computation Offloading and Service Caching in Digital Twin Edge Networks

LINGXIAO CHEN[1], QIANGQIANG GU[1], KAI JIANG[2], (Student Member, IEEE), AND LIANG ZHAO[1], (Member, IEEE)

[1]Hubei Key Laboratory of Intelligent Vision Based Monitoring for Hydroelectric Engineering, College of Computer and Information Technology, China Three Gorges University, Yichang 443002, China
[2]School of Cyber Science and Engineering, Wuhan University, Wuhan 430000, China

Corresponding author: Liang Zhao (zhaoliang@ctgu.edu.cn)

This work was supported by the Scientific Research Fund of Hubei Provincial Department of Education under Grant Q20221202.

ABSTRACT The combination of Mobile Edge Computing (MEC) and Digital Twin (DT) is anticipated to enhance the quality of mobile application services in the 6G era. However, current research often overlooks service caching and task dependency, which may deteriorate system performance. Moreover, Edge Servers (ESs) have limited computing resources and caching capacities, which require collaboration to meet user demands. To address these challenges, we propose a DT-empowered MEC architecture that supports Mobile Users (MUs) offloading dependency-aware tasks, while considering service caching and edge collaboration. The objective is to jointly optimize computation offloading and resource allocation to minimize the system's energy consumption. Hence, this problem can be formulated as a Mixed Integer Non-linear Programming (MINLP) problem and addressed by utilizing the Asynchronous Advantage Actor-Critic (A3C)-based method. Extensive simulation results demonstrate that our approach outperforms other benchmark algorithms under various scenarios, significantly reducing energy consumption.

INDEX TERMS Mobile edge computing, service caching, digital twin, deep reinforcement learning, task dependency.

I. INTRODUCTION

The rapid development of mobile applications, including autonomous driving, face recognition, and augmented reality, has been facilitated by the emergence of B5G/6G technology and intelligent terminals of Mobile Users (MUs). In order to address the challenge of reducing response delay and enhancing the user experience, a flexible paradigm called Mobile Edge Computing (MEC) has been introduced. It is capable of leveraging computing power at the edge network, decreasing task transmission delays and costs while simultaneously improving the Quality of Service (QoS) [1], [2], [3], [4].

Digital Twin (DT) technology connects and synchronizes the digital model of a physical entity or system with its actual operating state in real-time [5], [6]. With the revival of artificial intelligence, Reinforcement Learning (RL), including multi-agent RL and Deep RL (DRL), has recently been utilized to improve offloading efficiency in MEC [7]. However, MUs' limited storage and computing capacity hinder their ability to store substantial data and train neural networks. Integrating MEC and DT offers a practical solution to this problem [8]. More precisely, DT can gather a lot of physical layer data for neural network training, which will assist MUs in making optimal decisions.

Previous works about MEC have primarily focused on computation offloading, neglecting the significance of service caching and task dependency [9], [10], [11]. However, the performance and feasibility of task offloading are considerably affected by service caching and task dependency. Service caching indicates pre-caching application services and relevant databases on edge servers (ESs) so that they can handle corresponding computing tasks, reducing delay

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

as well as energy consumption [12], [13], [14]. Additionally, numerous computation-intensive tasks consist of multiple interdependent subtasks, where the processing of the current subtask is reliant on the completion of the preceding subtask [15], [16], [17].

Meanwhile, the aforementioned research was conducted in relatively simple application scenarios. This paper seeks to minimize the energy consumption of the system by addressing the problem of intelligent offloading for dependency-aware tasks. Furthermore, edge collaboration and service caching are considered, and resource allocation and computation offloading strategies are jointly optimized to handle time-varying network conditions. This paper's notable contributions are summarized below:

1) A DT-empowered MEC system architecture is proposed. By collecting system information, DT can train neural networks and improve training accuracy.
2) Taking service caching and task dependency into account, we can decrease the system's energy consumption by jointly optimizing offloading and resource allocation. This challenge is addressed by formulating it as a Mixed Integer Non-Linear Programming (MINLP) problem and presenting an A3C-based approach.
3) Simulation results demonstrate that in different scenarios, the proposed A3C-based method outperforms other benchmark schemes.

A review of related work is presented in Section II of this article, while Section III outlines the system model. Following that, Section IV offers an A3C-based algorithm as a potential solution to the problem at hand. Section V summarizes the simulation results, with the conclusion presented in Section VI.

II. RELATED WORK

To alleviate the workload pressure of limited MU resources, the research around computation offloading in MEC has aroused widespread interest over the past several years [18], [19]. In [20], the authors exploited a novel edge computing platform point to minimize user delay based on the wireless access. The authors in [21] utilized cloud-edge collaboration and vehicle-to-vehicle offloading in vehicular edge networks, aiming to maximize the system utility while satisfying the delay. In [22], the authors endeavored to reduce the energy consumption of MUs by optimizing the computing resources of ESs, in addition to taking computation offloading decisions into account. Furthermore, the authors in [23] investigated the end resource management challenge for application in an edge environment, considering energy efficiency and fairness.

In practice, however, ESs can only use existing services to accommodate specific computing tasks. The above works are unrealistic, since most assume that all tasks can be accommodated by an ES. In [24], the authors assumed the MEC server had cached the calculation results of high popularity before the user requested a computing task. The authors in [25] proposed that an ES cannot cache all services to support users, and computing tasks cannot be executed normally if the associated services are not cached. However, the associated task service type may not be cached because of the ES's restricted storage space, which leads to task failure. Besides, frequent service updates can result in higher operating costs than task execution and affect system stability [26]. Therefore, this problem should be solved by considering the collaboration between ESs. In [27], the authors considered computation offloading and service caching in a coordinated MEC platform. Fully leveraging the storage and computing capabilities, the authors in [28] not only consider workload scheduling and service caching but also emphasize the importance of edge node collaboration in reducing the overall workload of a MEC system.

However, most of the above studies focus on offloading independent tasks and rarely consider the correlation between tasks. The fact is that a mobile application may consist of multiple related tasks. Hence, in MEC scenarios, it is more challenging and practical to study the correlation between tasks for task offloading. In [29], the authors investigated a single-ES system that can optionally cache previously downloaded applications for future reuse. The authors in [30] considered how to find a feasible offloading scheme for dependency-aware tasks under limited cache resources, aiming to minimize the generation time. In [31], the author investigated the scheduling decision of an application composed of multiple related tasks in a cloud computing system and reduced the overall application execution cost. In [32], offloading assignments for dependency-aware tasks are investigated to attain the ideal offloading strategy based on the system cost.

Furthermore, as DT technology can evaluate the current system state in real-time and predict its future performance, it is enabled to play a comprehensive analysis and decision-making role in MEC scenarios. The authors in [33] proposed a deep learning (DL) architecture where DT can obtain user association schemes and the variation of real networks in real-time to train the DL algorithm offline. In [34], the author utilized DT technology to realize artificial intelligence in a vehicle edge computing network to promote the utilization of the ES' resources. The authors in [5] considered intelligently offloading tasks with the assistance of DT, considering collaboration between ESs in this system.

A DT-empowered MEC architecture that takes into account edge collaboration and service caching is proposed in this study. Separate from the majority of earlier research, we put the spotlight on the more challenging dependency-aware tasks as opposed to independent tasks.

III. SYSTEM MODEL

Detailed information regarding the network architecture is presented in this section. Following that, we delve into the

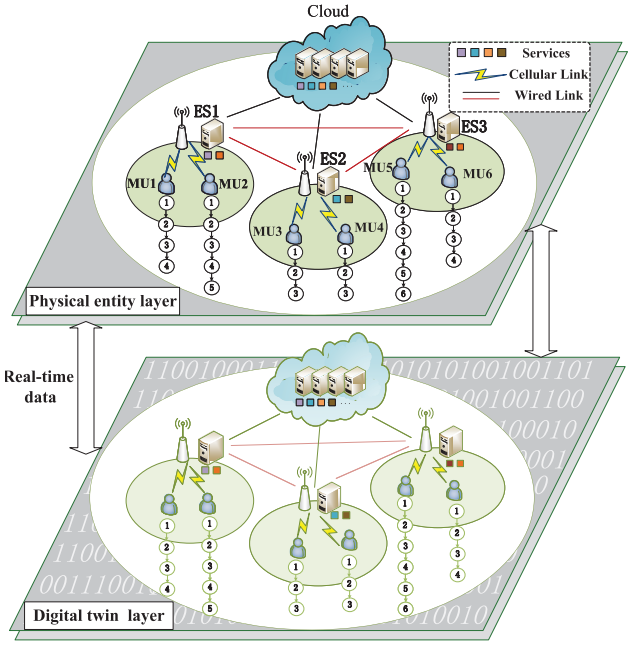

FIGURE 1. The Architecture of digital twin network.

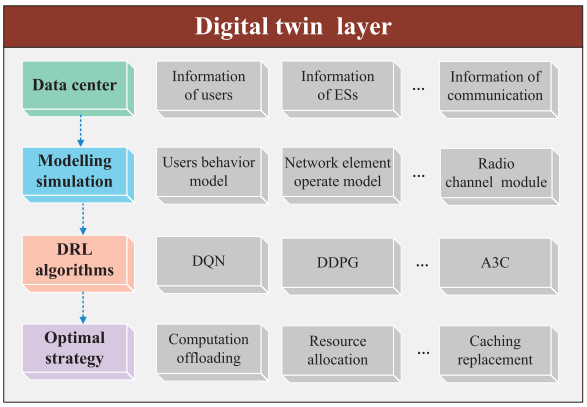

FIGURE 2. Structure of digital twin layer.

communication model and computation model. To conclude, we propose relevant optimization objectives.

A. NETWORK ARCHITECTURE

Fig. 1 illustrates a DT-empowered MEC architecture, which comprises a cloud server, multiple MUs, and ESs. This architecture is comprised of two layers, namely the physical entity layer and the DT layer. The former includes N ESs and I MUs, denoted by the sets $\mathcal{N} = \{1, 2, \ldots, N\}$ and $\mathcal{I} = \{1, 2, \ldots, I\}$, respectively. These ESs can receive task offloading requests from the MUs that are randomly distributed in the wireless coverage area. Additionally, both edge devices and terminals in the DT layer are equipped with corresponding sensors to collect relevant information, such as hardware configurations and network status.

As shown in Fig. 2, the data center in DT stores a large amount of data, including information about MUs, ESs, and communication environments. The data is utilized to create function modules for digital modeling, including modules for user behavior and radio channel environment. Through these models and big data, it is possible to utilize AI algorithms like DQN and DDPG to obtain the optimal strategy. Based on the above, DT collects a large amount of information from the physical entity layer. After receiving the user request and task information, the AI algorithm is used to determine which offloading scheme the MU adopts, and finally the offloading scheme is sent to the MU to realize the task's intelligent offloading. As a result, DT can constantly communicate with the physical layer to make valid real-time decisions.

In the proposed system, each MU i generates a task that is composed of multiple subtasks, which are dependent on each other. The set of subtasks is denoted by $\mathcal{J} = \{1, 2, \ldots, J\}$, where the output of subtask j' is utilized as the input for subtask j. Consequently, subtask j can only begin computation once subtask j' has been completed. The subtask information is characterized by various parameters, such as $c_{i,j}$ which specifies the aggregate number of CPU cycles needed to complete the subtask, $d_{i,j}$ which indicates the subtask's data size, and $t_{i,j}^{max}$ which represents the maximum acceptable delay for the subtask j.

Each ES n is restricted by its caching capacity, which limits the number of services it can cache. Specifically, only a subset of subtask types requires storage space h_q to cache corresponding services, and we denote this set as $\mathcal{Q}$. For each subtask type $q \in \mathcal{Q}$, let $\varphi_{n,q}$ denote whether ES n caches the corresponding service ($\varphi_{n,q} = 1$ if yes, and $\varphi_{n,q} = 0$ otherwise). The cloud has abundant caching capacity and can cache all services. Additionally, assuming that collaboration among ESs can enable distributed caching and enhance the utilization of available resources.

B. COMMUNICATION MODEL

In general, there is a wireless connection between MU and the ES. However, it is assumed that all tasks cannot be performed on the user's equipment and must be sent to the local ES first. The process of optimizing offloading computing tasks to the ES and returning the final result to MU is not being considered in this paper. Thereby, the wireless transmission process is ignored. In this study, communication between servers is facilitated by optical fiber wired connections. This encompasses the communication between an ES and the cloud server, as well as the interaction between various ESs. The data transfer speed between different ESs is represented as r_{es}, while r_j^{cloud} refers to the data transfer speed between each ES and the cloud server. Besides, $P_n^{trs}(J/bit)$ indicates the energy consumption per bit of data transmission between different ESs, while $P_{cloud}(J/bit)$ indicates the energy consumption per bit of data transmission between the ES and the cloud server. Thus, the transmission delay and energy consumption for ES-to-ES communication are calculated as.

$$t_{i,n,j}^{trs} = \frac{d_{i,j}}{r_{es}}, \tag{1}$$

$$e_{i,n,j}^{trs} = P_k^{trs} d_{i,j}, \tag{2}$$

Subsequently, the transmission delay and energy consumption between the cloud and ESs can be expressed as follows

$$t_{i,n,j}^{cloud} = \frac{d_{i,j}}{r_j^{cloud}}, \tag{3}$$

$$e_{i,n,j}^{cloud} = P_{cloud} d_{i,j}, \tag{4}$$

C. COMPUTATION MODEL

1) ES EXECUTION MODEL

The computing capacity offered by ES n to subtask j is represented by $f_{i,n,j}$, and the delay required to execute subtask j on ES n is expressed as follows

$$t_{i,n,j}^{comp} = \frac{c_{i,j}}{f_{i,n,j}}. \tag{5}$$

Next, the energy consumption on ES n can be calculated as follows:

$$e_{i,n,j}^{comp} = P_n^{comp} t_{i,n,j}^{comp}, \tag{6}$$

where P_n^{comp} indicates the computing power of ES n.

2) CLOUD EXECUTION MODEL

For tasks that require high computing capacities, they can be offloaded to cloud computing centers to execute through the core network. This allows the utilization of their powerful computing resources to handle tasks that are too complex to be processed by the terminal and ESs. Given the high downlink rate and small computation results, the computing delay $t_{i,n,j}^{cloud}$ on the cloud server and the transmission delay $e_{i,n,j}^{cloud}$ of returning results are not considered in this paper.

Binary decision variables are defined for each subtask to determine whether it should be carried out on the ES or the cloud. Specifically, $a_{i,n,j} = 0, 1$ indicates whether subtask j is carried out on the ES n, while $a_{i,c,j} = 0, 1$ indicates whether subtask j is carried out on the cloud. Moreover, the variables $a_{i,n',j'} = 1$ and $a_{i,c,j'} = 1$ represent the previous subtask j' being carried out on the ES n' or on the cloud, respectively.

Considering the location of the subtask execution and the input data source, the following situations are taken into account for subtask j of MU i.

1) Performing a subtask at the same location as its input data source results in further transmission delay or energy consumption, which occurs in two scenarios: (a) when the subtask is carried out in an ES and its input data is also sourced from the same ES ($a_{i,n',j'} = 1, a_{i,n,j} = 1, n' = n$), or (b) when both the current subtask and the preceding related subtask are carried out in the cloud ($a_{i,c,j'} = 1, a_{i,c,j} = 1$). Performing two sequential subtasks in the same location results in increased transmission delay or energy consumption.

2) Sequential subtask execution in different locations does not result in further transmission delay or energy consumption. This holds true when: (c) the subtask is carried out in the cloud and its input is caused by an ES $\left(a_{i,n,j'} = 1, a_{i,c,j} = 1\right)$, (d) the subtask is carried out in an ES and its input is caused by the cloud $\left(a_{i,c,j'} = 1, a_{i,n,j} = 1\right)$, or (e) the subtask is carried out in one ES and its input is caused by another ES $\left(a_{i,n',j'} = 1, a_{i,n,j} = 1, n' \neq n\right)$. Conversely, when the subtask execution location and input data source are the same, there will be an increase in transmission delay or energy consumption.

TABLE 1. Notations and definition.

Notations	Definition
$\mathcal{I}$	Number of MUs
$\mathcal{N}$	Number of ESs
$\mathcal{J}$	Number of subtasks
$\mathcal{Q}$	Number of subtask types
$c_{i,j}$	Total number of required CPU cycles for finishing the subtask i
$d_{i,j}$	Data size of the subtask i
$t_{i,j}^{max}$	The delay constraint of the subtask i
$\varphi_{n,q}$	Whether ES n caches the related service type q
r_{es}	The data transfer speed between two different ESs
r_j^{cloud}	The data transfer speed between each ES and the cloud
P_n^{trs}	The energy consumption for transmitting each bit of data between different ESs
P_n^{comp}	The computing power of ES n
P_{cloud}	The energy consumption for transmitting each bit of data between the cloud server and the ES
$f_{i,n,j}$	Allocated computing resources for subtask j by ES n
$a_{i,n,j}$	The decision of whether MU i should offload subtask j to ES n
$a_{i,n,j'}$	The decision of whether MU i should offload previous subtask j to ES n.
$a_{i,c,j}$	The decision of whether MU i should offload the subtask j to the cloud
$a_{i,c,j'}$	The decision of whether MU i should offload the previous subtask j to the cloud

According to the preceding, expressing the finish delay of subtask j formed by MU i can be done as follows:

$$t_{i,j}^{fin} = \begin{cases} t_{i,j'}^{fin} + t_{i,n,j}^{comp}, & a_{i,n',j'} = 1, a_{i,n,j} = 1, n' = n, \\ t_{i,j'}^{fin} + t_{i,n,j}^{trs} + t_{i,n,j}^{comp}, & a_{i,n',j'} = 1, a_{i,n,j} = 1, n' \neq n, \\ t_{i,j'}^{fin} + t_{i,n,j}^{cloud} + t_{i,n,j}^{comp}, & a_{i,c,j'} = 1, a_{i,n,j} = 1, \\ t_{i,n'}^{fin}, & a_{i,c,j'} = 1, a_{i,c,j} = 1, \\ t_{i,j'}^{fin} + t_{i,n,j}^{cloud}, & a_{i,n,j'} = 1, a_{i,c,j} = 1, \end{cases} \tag{7}$$

where $t_{i,j'}^{fin}$ indicates the finish delay of subtask j' of MU i. Hence, the maximum finish delay of MU i is defined as $T_i = \max\left\{t_{i,j}^{fin}, j \in J\right\}$. In addition, the energy consumption of subtask j is

$$e_{i,j}^{fin} = \begin{cases} e_{i,j'}^{fin} + e_{i,n,j}^{comp}, & a_{i,n',j'} = 1, a_{i,n,j} = 1, n' = n, \\ e_{i,j'}^{fin} + e_{i,n,j}^{trs} + e_{i,n,j}^{comp}, & a_{i,n',j'} = 1, a_{i,n,j} = 1, n' \neq n, \\ e_{i,j'}^{fin} + e_{i,n,j}^{cloud} + e_{i,n,j}^{comp}, & a_{i,c,j'} = 1, a_{i,n,j} = 1, \\ e_{i,j'}^{fin}, & a_{i,c,j'} = 1, a_{i,c,j} = 1, \\ e_{i,j'}^{fin} + e_{i,n,j}^{cloud}, & a_{i,n,j'} = 1, a_{i,c,j} = 1, \end{cases} \tag{8}$$

Accordingly, the system energy consumption for finishing total subtasks of MU i can be represented as $E_i = \max\left\{e_{i,j}^{fin}, j \in J\right\}$.

D. PROBLEM FORMULATION

Minimizing the energy consumption of the system while fulfilling the delay demands of each subtask is our objective. Given the system model described above, the corresponding problem can be described as

$$\min_{\mathbf{a},\mathbf{f}} \quad \sum_{j=1}^{J} E_i \tag{9}$$

$$s.t. \quad a_{i,n,j} = \{0,1\}, \forall i \in \mathcal{I}, \forall n \in \mathcal{N}, \forall j \in \mathcal{J} \tag{9a}$$

$$a_{i,c,j} = \{0,1\}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J} \tag{9b}$$

$$\sum_{j \in J} a_{i,n,j} + a_{i,c,j} = 1, \forall i \in \mathcal{I}, \forall n \in \mathcal{N} \tag{9c}$$

$$\sum_{q \in Q} \varphi_{n,q} h_q \leq C_n, \forall n \in \mathcal{N} \tag{9d}$$

$$\sum_{j \in J} f_{i,n,j} \leq f_n^{edge}, \forall n \in \mathcal{N} \tag{9e}$$

$$t_{i,j}^{fin} - t_{i,j'}^{fin} \leq t_{i,j}^{max}, \forall i \in \mathcal{I}, \forall j \in \mathcal{J}, \forall j' \in \mathcal{J} \tag{9f}$$

$$T_i \leq T_i^{max}, \forall i \in \mathcal{I}, \tag{9g}$$

where (9a) and (9b) are binary decision variables that represent offloading tasks to the ES n and the cloud, respectively; (9c) indicates that each subtask has the option to be executed either on the ES n or the cloud. To ensure that the total amount of cached services does not be exceed the storage capacity C_n of the ES n, (9d) is introduced. The available computing resources f_n^{edge} given by the ES n is guaranteed not to be surpassed by the allocated computing resources for MUs through (9e). Additionally, the delay for each subtask and total subtasks are respectively restricted by (9f) and (9g). It is important to note that T_i^{max} expresses the maximum allowable delay for executing all subtasks.

So as to decrease the system's energy consumption, efficient offloading and resource allocation decisions are required. Nevertheless, $a_{i,n,j}$ and $a_{i,c,j}$ are binary discrete variables, while $f_{i,n,j}$ is a continuous variable. That is to say, this problem can be described as a MINLP problem. Although MOSEK, Gurobi, and other solvers have many mature algorithms for solving MINLP problems, they need to assume that the environment is static and will not change due to decision-making actions, that is, actions will not affect the environment. They can only obtain the optimal solution in a certain state and cannot solve the sequential decision-making problem. The RL algorithm is self-adaptive and continuously improves its decision-making strategy through interaction with the environment, gradually approaching the optimal strategy. This study considers a system model where the tasks consist of interdependent subtasks. Specifically, the succeeding subtask cannot commence until the previous one has been completed, and the choices made for each subtask will influence subsequent decisions. As such, this issue represents a standard sequential decision-making problem.

To summarize, traditional RL algorithms like Q-learning and SARSA are inadequate for the MDP problem because it involves both continuous and discrete actions. However, A3C is a DRL method that is based on the actor-critic framework and can handle such problems. It interacts with the environment through multiple threads at the same time, and trains the neural networks of multiple workers asynchronously. Therefore, the A3C algorithm is presented to address this issue.

IV. PROPOSED A3C BASED ALGORITHM

The fundamental elements of our system will now be explored in this section. Afterwards, we will introduce a DRL method customized to tackle the issue previously described.

A. DEFINITION OF THE STATE, ACTION AND REWARD

Each MU generates a series of dependent subtasks, which can be offloaded not only to associated ESs but also collaborative ESs or the cloud via associated ESs. The computation results are ultimately returned to the MU after completing all subtasks. The Markov Decision Process (MDP) can model the environment in which intelligence is placed. Here, we formally represent the optimization of the computation offloading process as an MDP. In this MDP, the agent can repeatedly interact with the unknown environment and make optimal decisions adaptively. Three important factors in MDP are now defined as shown below.

1) **State space:** The network environment can represent the system state. Thereby, the state at time slot t can be indicated as $s(t) = \{\mathcal{L}(t), \mathcal{A}(t)\}$, $\mathcal{L}(t)$ refers to caching service type, idle caching storage and computing capacity of the ES at time slot t. $\mathcal{A}(t) = \left\{a_{i,n,j}^{t-1}, a_{i,c,j}^{t-1} \mid i \in \mathcal{I}, n \in \mathcal{N}, j \in \mathcal{J}\right\}$, denotes the current subtask's input data source.

2) **Action space:** It is utilized to represent related decisions at each time slot. According to our proposed system, the action space can be indicated as $a_t = \{\mathcal{A}(t), \mathcal{F}(t)\}$. The offloading decision vector can be indicated as $\mathcal{A}(t) = \left\{a_{i,n,j}^{t}, a_{i,c,j}^{t} \mid i \in \mathcal{I}, n \in \mathcal{N}, j \in \mathcal{J}\right\}$, and the computing resource allocation is indicated as $F(t) = \left\{f_{i,n,j}^{t} \mid i \in \mathcal{I}, n \in \mathcal{N}, j \in \mathcal{J}\right\}$.

3) **Reward Function:** The agent receives an immediate reward for each action taken based on the current state at each time step. Our goal is to reduce the energy consumption of the system while fulfilling the delay conditions of each subtask. Consequently, we have designed the reward function based on the corresponding optimization goal, which can be expressed as

$$R(s(t), a(t)) = -\sum_{i=1}^{I} e_{i,j}^{fin} - e_{i,j'}^{fin}(t), \tag{10}$$

where $\sum_{i=1}^{I} e_{i,j}^{fin} - e_{i,j'}^{fin}(t)$ indicates the total system's energy consumption.

B. DEEP REINFORCEMENT LEARNING

As a generic learning framework, RL contains both the environment, the agent, observation, action, and reward. In each episode, the agent monitors the environment's condition and interacts with it by taking actions, and receiving different levels of rewards. After continuous learning, the optimal strategy for maximizing the long-term expected cumulative reward will be acquired. Nevertheless, traditional RL approaches have difficulty handling high-dimensional state spaces or more complex continuous actions [35]. DRL has been widely used in a variety of applications in recent years, such as augmented reality, smart homes, intelligent manufacturing, etc. It represents Q-values by value function approximation instead of a Q-table, solving the problem that traditional RL cannot handle dimensional disasters.

In addition, most RL algorithms have slow training speeds. However, A3C is a DRL method that can utilize multiple threads to accelerate the learning process.

C. A3C-BASED ALGORITHM

By employing several threads to interact with the environment concurrently, the A3C enables multiple workers for asynchronous training of the neural network [36], [37]. The global neural network model in the A3C algorithm is responsible for storing and updating network parameters. Each worker thread carries out an action established on the current state and observes a reward, which will be transmitted back to the neural network for parameter adjustment, aiding the agent to learn and optimize its strategy. This is achieved by calculating the gradient of the neural network loss function.

The A3C algorithm also makes use of the actor-critic network framework, which serves two purposes. Firstly, the actor strategy network aims to learn and optimize the strategy. The actor network receives the state as input and generates the action probability distribution, improving the performance of the strategy. Secondly, the critic network manages measuring the performance of the actor in a particular state.

Then, the A3C process will be described. The DT system initiates the information gathering process from the immediate surroundings to deduce the current state s_t. Subsequently, guided by the policy function $\pi\left(a_t \mid s_t; \theta\right)$, the system undertakes a corresponding action a_t, and as a result, obtains a reward r_t before transitioning to the subsequent state s_{t+1}. The state value function $V\left(s_t; \theta_v\right)$, which is parameterized by θ_v, is mathematically represented by

$$\begin{aligned} V\left(s_t; \theta_v\right) &= E\left[G_t \mid s = s_t, \pi\right] \\ &= E\left[\sum_{k=0}^{\infty} \gamma^k r_{t+m} \mid s = s_t, \pi\right], \end{aligned} \tag{11}$$

where G_t represents the discounted return for the state s_t. The discount factor γ represents how future returns will affect the current system state, where $\gamma \in [0, 1]$.

The A3C algorithm adopts the m-step update method for parameter updating, which is faster than the one-step return method. The cumulative reward of m steps can be represented as

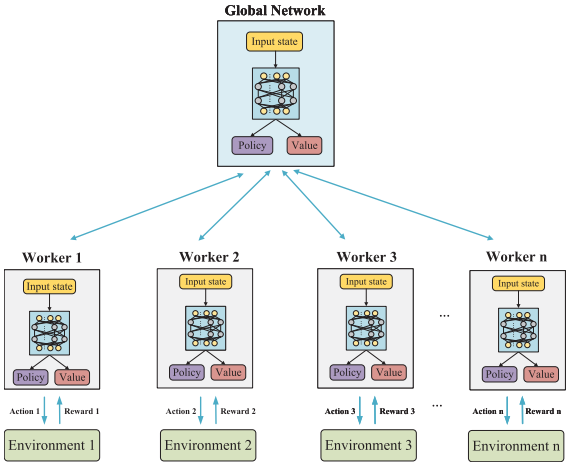

FIGURE 3. The architecture of A3C.

$$R_t = \sum_{i=0}^{m-1} \gamma^i r_{t+i} + \gamma^m V\left(s_{t+m}; \theta_v\right), \tag{12}$$

where r_{t+i} represents the immediate reward. m is upper-bounded by t_{max}. If it achieved the final state or performed the t_{max} action, both the value and policy functions will be updated.

Furthermore, to improve the efficiency and stability of training, the advantage function is introduced and defined as

$$\begin{aligned} A\left(s_t, a_t; \theta, \theta_v\right) = R_t - V\left(s_t; \theta_v\right) = \sum_{i=0}^{m-1} \gamma^i r_{t+i} \\ + \gamma^k V\left(s_{t+m}; \theta_v\right) - V\left(s_t; \theta_v\right), \end{aligned} \tag{13}$$

where the value function and the parameters of the policyare denoted by θ_v and θ, respectively. By comparing the action taken in the current state with the expected return and the average return, it can estimate the quality of an action and help the A3C-based algorithm adjust its policy.

Based on the above, the actor network's loss function is written as

$$f_\pi(\theta) = \log \pi\left(a_t \mid s_t; \theta\right)\left(R_t - V\left(s_t; \theta_v\right)\right) + \delta H\left(\pi\left(s_t; \theta\right)\right), \tag{14}$$

where the entropy term of strategy π can be represented by $H\left(\pi\left(s_t; \theta\right)\right)$, and δ can maintain the balance between the exploitation and exploration [38]. The critic network's loss function is written as

$$f_v\left(\theta_v\right) = \left(R_t - V\left(s_t; \theta_v\right)\right)^2. \tag{15}$$

Next, the policy parameter of actor network and critic network are updated, which can be expressed as

$$\begin{aligned} d\theta \leftarrow d\theta + \nabla_{\theta'} \log \pi\left(a_t \mid s_t; \theta'\right)\left(R_t - V\left(s_t; \theta_v\right)\right) \\ + \delta \nabla_{\theta'} H\left(\pi\left(s_t; \theta'\right)\right), \end{aligned} \tag{16}$$

Algorithm 1 A3C-Based Joint Computation Offloading and Resource Allocation Algorithm

Input: The initial state and related environment parameters;
Output: The optimal computation offloading and resource allocation action; **Iteration:**

repeat
  for each worker **do**
    Set gradients of two global networks: $d\theta = 0$, $d\theta_v = 0$;
    Synchronous parameters of each worker with global parameters: $\theta' = \theta$ and $\theta'_v = \theta_v$;
    Obtain the current system state s_t;
    for $t < t_{\max}$ **do**
      Perform an action a_t according to the policy $\pi(a_t \mid s_t; \theta)$;
      Reward with r_t and new state s_{t+1};
      $t \leftarrow t + 1$;
    end for
    $R = \begin{cases} 0, & \text{for terminal state} \\ V\left(s_t, \theta'_v\right), & \text{for non - terminal state.} \end{cases}$
    for $i \in \{t-1, \cdots, t_{\text{start}}\}$ **do**
      Update $R = r_t + \gamma R$;
      Obtain cumulative gradient wrt θ' by Eq. (16);
      Obtain cumulative gradient wrt θ'_v by Eq. (17);
    end for
    Asynchronous update θ and θ_v;
    $T \leftarrow T + 1$;
  end for
until $T > T_{\max}$

and

$$d\theta_v \leftarrow d\theta_v + \frac{\partial \left(R_t - V\left(s_t; \theta_v\right)\right)^2}{\partial \theta'_v}. \tag{17}$$

In light of the RMSProp optimization, the global parameter θ and θ_v are updated in an asynchronous manner [38], [39].

Algorithm 1 outlines the details of the A3C-based algorithm, and the optimal decisions are made by the agent based on this algorithm. Firstly, there is a global network and some workers composed of actors and critics, and each worker learns the recent parameters from the global network and obtains the present system state for each episode. It individually selects actions depending on the present strategy s_t and interacts with the environment to get reward r_t and achieve following the state s_{t+1}. Until the ultimate state is reached, this process is repeated. Next, each worker updates its actor and critic network according to the cumulative gradient function. In addition, the global network will automatically update its parameters after obtaining the uploaded parameters of each worker. In each episode until the last one, repeat the appeal training process. When the final time slot is reached, the algorithm converges, and the optimal strategy is obtained.

TABLE 2. Simulation parameters.

Parameter	Value
Number of ESs	3
Number of MUs	[6,10]
Number of subtask types	10
Number of subtasks	[3,8]
The data size of subtask	$[1, 4]$ Mbit
Computing resources of ES	$[4, 8]$ GHz
Caching capacity of ES	$[10, 60]$ GB
Service storage demand	$[1, 4]$ GB

V. NUMERICAL RESULTS

A. PARAMETERS SETTINGS

To assess the effectiveness of the proposed method, we carried out experiments in a dynamic MEC scenario that comprises a cloud and 3 ESs along with 6 MUs. The subtasks generated by MUs are of 10 different types, and their sizes fall between 1 and 4 Mbit. Each ES has a computing capacity ranging from 4 to 8 GHz and caching capacity between 10 and 60 GB. The storage demand for each service ranges from 1 to 4 GB. $P_{cloud}(j/bit) = 2 \times 10^{-7}(J/bit)$ represents the energy consumed per bit of data transmitted from the ES to the cloud server, while $P_n^{trs} = 3 \times 10^{-8}(J/bit)$ represents the energy consumed per bit of data transmitted between ESs.

Furthermore, we adopt a fully connected neural network for our actor-critic network with two hidden layers, each consisting of 128 hidden units. The experience playback buffer size is set to 5000, with a fixed mini-batch of 64 sample transfer tuples. In addition, the actor and critic learning rates are 0.001 and 0.01, respectively.

To fully evaluate the A3C algorithm, the following benchmark schemes are compared with it, along with DDPG-based and Greedy algorithms:

- **DDPG-based**: This algorithm uses deep deterministic strategies to obtain optimal decisions.
- **Greedy**: The agent will adopt the strategy with the lowest energy consumption for each subtask. Furthermore, the computing resources assigned to each subtask must satisfy its delay constraint.
- **Cloud Execution (CE)**: All subtasks are carried out in the cloud. Therefore, ESs do not participate in the execution of subtasks, and its computing resources are not used.
- **Random Offloading (RO)**: Subtasks are randomly assigned to the ES that stores the relevant service. If the service is not cached by any ES, it is offloaded to the cloud.

B. THE CONVERGENCE PERFORMANCE

We will compare the A3C and DDPG algorithms based on average reward and analyze their differences in this section. By contrasting the convergence performance of the two algorithms in Fig. 4, it is evident that the convergence curve of the A3C-based algorithm climbs rapidly within the

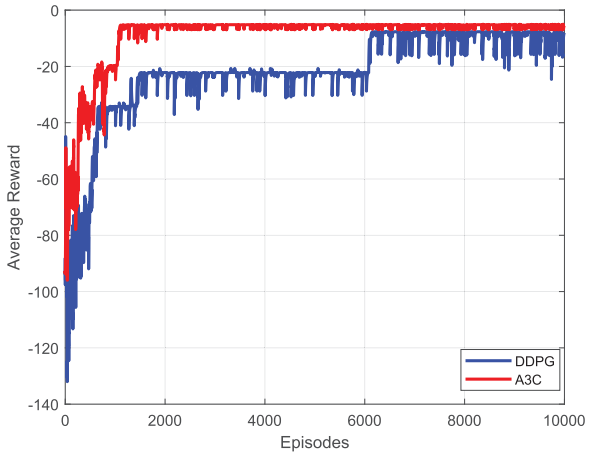

FIGURE 4. Convergence of the A3C-based and DDPG-based algorithms.

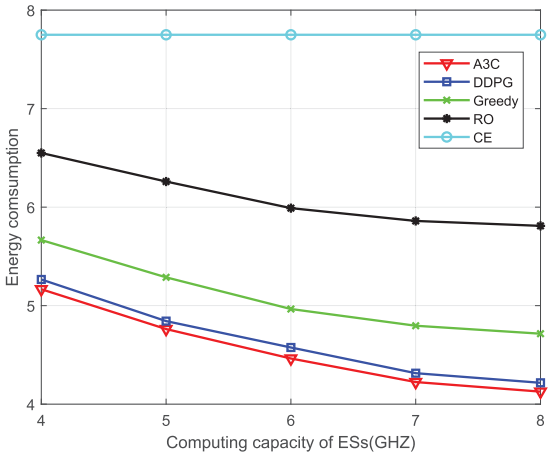

FIGURE 6. Energy consumption versus the computing resources of ESs.

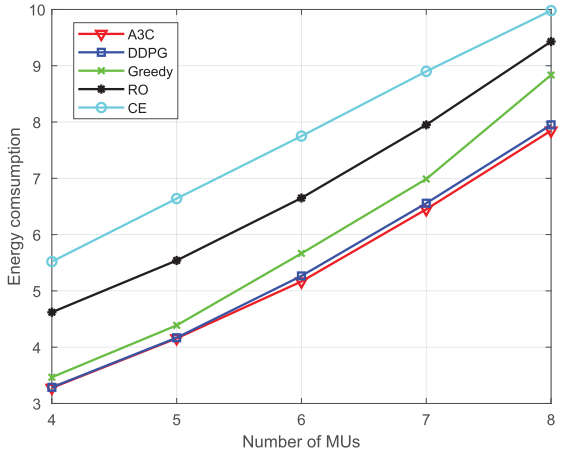

FIGURE 5. Energy consumption versus the number of MUs.

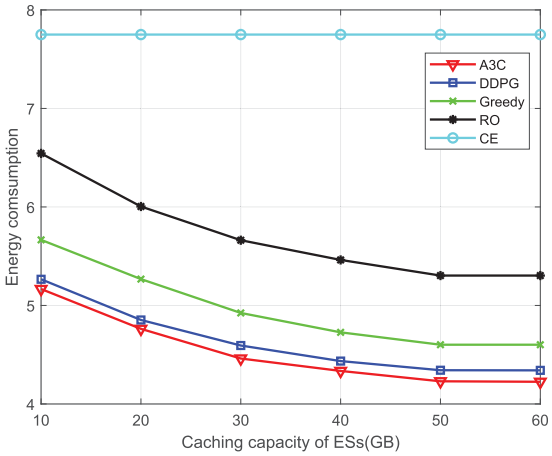

FIGURE 7. Energy consumption versus the caching capacity of ESs.

first 1500 training episodes and reaches a relatively stable value. Furthermore, we have noticed that the ultimate convergence outcome obtained via the A3C-based algorithm closely approximates that of the DDPG-based algorithm. However, the convergence speed of the former is significantly better than that of the latter. This discrepancy is attributed to the implementation of multi-threading in the A3C-based algorithm, which allows for greater interaction with the environment, thereby resulting in enhanced training efficacy.

C. PERFORMANCE COMPARISON

Additionally, we contrast the energy consumption performance of our proposed algorithm with the DDPG-based algorithm, Greedy, RO, and CE under different scenarios.

Fig. 5 exhibits the relationship between the energy consumption and the number of MUs. It is obvious that two DRL algorithms have the best performance. The main reason is that they can obtain the optimal decision by training the neural network with a large amount of data. Greedy algorithm tends to get the local optimal solution without considering the overall situation. Hence, our proposed method outperforms the Greedy algorithm. With an increase in the number of MUs, the energy consumption of all methods grows proportionally by generating more computing tasks, leading to more intense competition for computing resources.

Fig. 6 exhibits the relationship between the energy consumption and the computing capabilities of ESs. All schemes, except for CE, exhibit a decline in energy consumption as the number of ESs increases, which reduces the computing delay and, consequently, the energy consumption. In contrast, CE remains invariant as the computing resources of the ESs remain unutilized. Notably, the A3C-based algorithm outperforms the DDPG-based algorithm in terms of energy consumption reduction, owing to their adeptness in gathering global information and efficiently allocating computing resources.

The relationship between the energy consumption and the caching capacity of ESs is depicted in Fig. 7. For all schemes except the CE, energy consumption falls as caching capacity rises. This is because the ES can store additional services, providing more options for the offloading of subtasks. Some subtasks can be offloaded to collaborative ESs, and the cloud

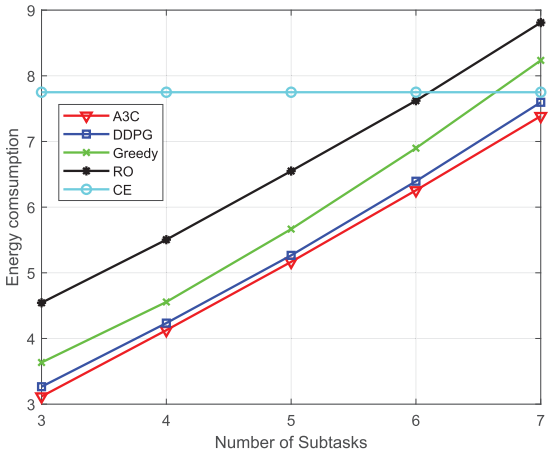

FIGURE 8. Energy consumption versus the number of subtasks.

can be offloaded to the local ES, thus reducing transmission energy consumption. For the CE scheme, energy consumption is not affected by the caching capacity of ESs because all the subtasks are carried out in the cloud rather than on the ES.

Fig. 8 illustrates the relationship between the energy consumption and the number of subtasks. All schemes exhibit a uniform energy consumption trend, with the exception of CE, wherein sequential subtasks are carried out in the cloud, incurring no additional energy expenditure. Notably, as the number of subtasks escalates to seven, both RO and Greedy approaches manifest higher energy consumption compared to CE. The superior energy efficiency of the A3C-based algorithm can be attributed to its aptitude for acquiring optimal decisions for these subtasks.

Overall, the proposed method outperforms other benchmark schemes in various scenarios, confirming its effectiveness in reducing energy consumption.

VI. CONCLUSION

This paper proposes a DT-empowered MEC system architecture considering service caching and task dependency together. Firstly, we formulate the original optimization problem as a MINLP. Next, an A3C-based solution is developed to decrease energy consumption by acquiring the optimal policy. Finally, the proposed method outperforms other benchmarks under different scenarios, as demonstrated by the simulation results. However, our study only considers a sequential dependency-aware task execution model, and further research is needed to explore task offloading under a more general model. In fact, studying task offloading under a general dependency-aware task model would have significant practical value. Furthermore, extended reality (XR) and holographic communication are the current research frontiers and will become one of the important applications of the future 6G communication network. In the following research, this aspect will be further considered.

REFERENCES

[1] Y. Liu, S. Xie, and Y. Zhang, "Cooperative offloading and resource management for UAV-enabled mobile edge computing in power IoT system," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12229–12239, Oct. 2020.

[2] S. Bebortta, D. Senapati, C. R. Panigrahi, and B. Pati, "Adaptive performance modeling framework for QoS-aware offloading in MEC-based IIoT systems," *IEEE Internet Things J.*, vol. 9, no. 12, pp. 10162–10171, Jun. 2022.

[3] H. Zhou, Z. Zhang, D. Li, and Z. Su, "Joint optimization of computing offloading and service caching in edge computing-based smart grid," *IEEE Trans. Cloud Comput.*, vol. 11, no. 2, pp. 1122–1132, Apr./Jun. 2022.

[4] X. Shang, Y. Huang, Y. Mao, Z. Liu, and Y. Yang, "Enabling QoE support for interactive applications over mobile edge with high user mobility," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 1289–1298.

[5] T. Liu, L. Tang, W. Wang, Q. Chen, and X. Zeng, "Digital-twin-assisted task offloading based on edge collaboration in the digital twin edge network," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1427–1444, Jan. 2022.

[6] X. Lin, J. Wu, J. Li, W. Yang, and M. Guizani, "Stochastic digital-twin service demand with edge response: An incentive-based congestion control approach," *IEEE Trans. Mobile Comput.*, vol. 22, no. 4, pp. 2402–2416, Apr. 2023, doi: 10.1109/TMC.2021.3122013.

[7] H. Zhang, M. Huang, H. Zhou, X. Wang, N. Wang, and K. Long, "Capacity maximization in RIS-UAV networks: A DDQN-based trajectory and phase shift optimization approach," *IEEE Trans. Wireless Commun.*, vol. 22, no. 4, pp. 2583–2591, Apr. 2023.

[8] B. Fan, Y. Wu, Z. He, Y. Chen, T. Q. S. Quek, and C. Xu, "Digital twin empowered mobile edge computing for intelligent vehicular lane-changing," *IEEE Netw.*, vol. 35, no. 6, pp. 194–201, Nov. 2021.

[9] G. Guo and J. Zhang, "Energy-efficient incremental offloading of neural network computations in mobile edge computing," in *Proc. IEEE GLOBECOM*, Dec. 2020, pp. 1–6.

[10] Y. Dai, D. Xu, S. Maharjan, G. Qiao, and Y. Zhang, "Artificial intelligence empowered edge computing and caching for Internet of Vehicles," *IEEE Wireless Commun.*, vol. 26, no. 3, pp. 12–18, Jun. 2019.

[11] Z. Song, Y. Liu, and X. Sun, "Joint task offloading and resource allocation for NOMA-enabled multi-access mobile edge computing," *IEEE Trans. Commun.*, vol. 69, no. 3, pp. 1548–1564, Mar. 2021.

[12] H. Zhou, T. Wu, H. Zhang, and J. Wu, "Incentive-driven deep reinforcement learning for content caching and D2D offloading," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 8, pp. 2445–2460, Aug. 2021.

[13] J. Yan, S. Bi, L. Duan, and Y. A. Zhang, "Pricing-driven service caching and task offloading in mobile edge computing," *IEEE Trans. Wireless Commun.*, vol. 20, no. 7, pp. 4495–4512, Jul. 2021.

[14] G. Zheng, C. Xu, H. Long, and X. Zhao, "MEC in NOMA-HetNets: A joint task offloading and resource allocation approach," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Mar. 2021, pp. 1–6.

[15] H. Liao, X. Li, D. Guo, W. Kang, and J. Li, "Dependency-aware application assigning and scheduling in edge computing," *IEEE Internet Things J.*, vol. 9, no. 6, pp. 4451–4463, Mar. 2022.

[16] M. Mehrabi, S. Shen, V. Latzko, Y. Wang, and F. H. P. Fitzek, "Energy-aware cooperative offloading framework for inter-dependent and delay-sensitive tasks," in *Proc. GLOBECOM IEEE Global Commun. Conf.*, Dec. 2020, pp. 1–6.

[17] W. He, L. Gao, and J. Luo, "A multi-layer offloading framework for dependency-aware tasks in MEC," in *Proc. IEEE Int. Conf. Commun.*, Jun. 2021, pp. 1–6.

[18] H. Zhou, T. Wu, X. Chen, S. He, D. Guo, and J. Wu, "Reverse auction-based computation offloading and resource allocation in mobile cloud-edge computing," *IEEE Trans. Mobile Comput.*, early access, Jul. 18, 2022, doi: 10.1109/TMC.2022.3189050.

[19] H. Zhou, M. Li, N. Wang, G. Min, and J. Wu, "Accelerating deep learning inference via model parallelism and partial computation offloading," *IEEE Trans. Parallel Distrib. Syst.*, vol. 34, no. 2, pp. 475–488, Feb. 2023.

[20] Z. Wang, G. Xue, S. Qian, and M. Li, "CampEdge: Distributed computation offloading strategy under large-scale AP-based edge computing system for IoT applications," *IEEE Internet Things J.*, vol. 8, no. 8, pp. 6733–6745, Apr. 2021.

[21] J. Zhao, Q. Li, Y. Gong, and K. Zhang, "Computation offloading and resource allocation for cloud assisted mobile edge computing in vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 7944–7956, Aug. 2019.

[22] H. Li, H. Xu, C. Zhou, X. Lü, and Z. Han, "Joint optimization strategy of computation offloading and resource allocation in multi-access edge computing environment," *IEEE Trans. Veh. Technol.*, vol. 69, no. 9, pp. 10214–10226, Sep. 2020.
[23] Y. Kim, H. Lee, and S. Chong, "Mobile computation offloading for application throughput fairness and energy efficiency," *IEEE Trans. Wireless Commun.*, vol. 18, no. 1, pp. 3–19, Jan. 2019.
[24] S. Li, B. Li, and W. Zhao, "Joint optimization of caching and computation in multi-server NOMA-MEC system via reinforcement learning," *IEEE Access*, vol. 8, pp. 112762–112771, 2020.
[25] Z. Xu, L. Zhou, H. Dai, W. Liang, W. Zhou, P. Zhou, W. Xu, and G. Wu, "Energy-aware collaborative service caching in a 5G-enabled MEC with uncertain payoffs," *IEEE Trans. Commun.*, vol. 70, no. 2, pp. 1058–1071, Feb. 2022.
[26] V. Farhadi, F. Mehmeti, T. He, T. F. L. Porta, H. Khamfroush, S. Wang, K. S. Chan, and K. Poularakis, "Service placement and request scheduling for data-intensive applications in edge clouds," *IEEE/ACM Trans. Netw.*, vol. 29, no. 2, pp. 779–792, Apr. 2021.
[27] H. Zhou, Z. Zhang, Y. Wu, M. Dong, and V. C. M. Leung, "Energy efficient joint computation offloading and service caching for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Green Commun. Netw.*, vol. 7, no. 2, pp. 950–961, Jun. 2023.
[28] X. Ma, A. Zhou, S. Zhang, and S. Wang, "Cooperative service caching and workload scheduling in mobile edge computing," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 2076–2085.
[29] S. Bi, L. Huang, and Y. A. Zhang, "Joint optimization of service caching placement and computation offloading in mobile edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 7, pp. 4947–4963, Jul. 2020.
[30] G. Zhao, H. Xu, Y. Zhao, C. Qiao, and L. Huang, "Offloading dependent tasks in mobile edge computing with service caching," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Jul. 2020, pp. 1997–2006.
[31] S. Sundar and B. Liang, "Offloading dependent tasks with communication delay and deadline constraint," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, Apr. 2018, pp. 37–45.
[32] Y. Fan, L. Zhai, and H. Wang, "Cost-efficient dependent task offloading for multiusers," *IEEE Access*, vol. 7, pp. 115843–115856, 2019.
[33] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, "Deep learning for hybrid 5G services in mobile edge computing systems: Learn from a digital twin," *IEEE Trans. Wireless Commun.*, vol. 18, no. 10, pp. 4692–4707, Oct. 2019.
[34] K. Zhang, J. Cao, and Y. Zhang, "Adaptive digital twin and multiagent deep reinforcement learning for vehicular edge computing and networks," *IEEE Trans. Ind. Informat.*, vol. 18, no. 2, pp. 1405–1413, Feb. 2022.
[35] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jan. 2022.
[36] X. Ye, M. Li, F. R. Yu, P. Si, Z. Wang, and Y. Zhang, "MEC and blockchain-enabled energy-efficient Internet of Vehicles based on A3C approach," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2021, pp. 1–6.
[37] H. Zhou, Z. Wang, G. Min, and H. Zhang, "UAV-aided computation offloading in mobile-edge computing networks: A Stackelberg game approach," *IEEE Internet Things J.*, vol. 10, no. 8, pp. 6622–6633, Apr. 2023.
[38] J. Du, F. R. Yu, G. Lu, J. Wang, J. Jiang, and X. Chu, "MEC-assisted immersive VR video streaming over terahertz wireless networks: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9517–9529, Oct. 2020.
[39] H. Zhou, Z. Wang, H. Zheng, S. He, and M. Dong, "Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 3, pp. 1326–1338, May 2023.

LINGXIAO CHEN received the B.S. degree from the Chongqing University of Posts and Telecommunications. He is currently pursuing the master's degree with the College of Computer and Information, China Three Gorges University. His main research interests include mobile edge computing and deep reinforcement learning.

QIANGQIANG GU received the B.S. degree from Nanyang Normal University. He is currently pursuing the master's degree with the College of Computer and Information, China Three Gorges University. His main research interests include mobile edge computing and federated learning.

KAI JIANG (Student Member, IEEE) received the M.S. degree from China Three Gorges University, China, in 2021. He is currently pursuing the Ph.D. degree in cyberspace security with Wuhan University, China. His research interests include deep reinforcement learning, intelligent transportation, the Internet of Vehicles, and mobile edge computing.

LIANG ZHAO (Member, IEEE) received the B.Sc. degree in biomedical engineering from the Changchun University of Technology, in 2015, and the M.Sc. degree in circuits and systems and the Ph.D. degree in radio physics from Central China Normal University, Wuhan, China, in 2018 and 2021, respectively. He is currently a Lecturer with the College of Computer and Information Technology, China Three Gorges University. His main research interests include wireless sensor networks, the Internet of Things, and mobile edge computing.