Deep-Reinforcement-Learning-Based Joint Caching and Resources Allocation for Cooperative MEC

Wenqian Zhang, Guanglin Zhang, *Member, IEEE*, and Shiwen Mao, *Fellow, IEEE*

***Abstract*—The emergence of new applications has led to a high demand for mobile-edge computing (MEC), which is a promising paradigm with a cloud-like architecture deployed at the network edge to provide computation and storage services to mobile users (MUs). Since MEC servers have limited resources compared to the remote cloud, it is crucial to optimize resource allocation in MEC systems and balance the load among cooperating MEC servers. Caching application data for different types of computing services (CSs) at MEC servers can also be highly beneficial. In this article, we investigate the problem of hierarchical joint caching and resource allocation in a cooperative MEC system, which is formulated as an infinite-horizon cost minimization Markov decision process (MDP). To deal with the large state and action spaces, we decompose the problem into two coupled subproblems and develop a hierarchical reinforcement learning (HRL)-based solution. The lower layer uses the deep *Q* network (DQN) to obtain service caching and workload offloading decisions, while the upper layer leverages DQN to obtain load balancing decisions among cooperative MEC servers. The feasibility and effectiveness of our proposed schemes are validated by our evaluation results.**

***Index Terms*—Cooperative MEC servers, deep reinforcement learning (DRL), hierarchical reinforcement learning (HRL), joint caching and resources allocation, mobile-edge computing (MEC).**

I. Introduction

THE RAPID development of mobile devices with perception and communication capabilities leads to the exponential growth of computing-intensive and delay-sensitive applications, which may bring unprecedented challenges to mobile devices due to their limited computation ability and storage resources [1], [2]. Furthermore, the workload of computing services (CSs) generated by mobile devices usually suffers long and variable delays for workload transmission if it is executed at a centralized remote cloud server [3]. To this end, the edge intelligence (EI) and the sixth-generation (6G) wireless networks are expected to pave the way for high-quality service performance [4]. Although mobile-edge computing (MEC) technology is deployed at the proximity of mobile users (MUs) and has been envisioned as an extension of cloud computing to provide computing and storage services for MUs within its coverage area [5], [6], [7], the MEC servers usually can hardly withstand numerous offloading requests from MUs due to their limited service capabilities. Therefore, how to design intelligent workload offloading under the premise of considering the reasonable resource allocation is worthwhile to be investigated [8].

A unique benefit of MEC is that MEC servers can prestore the popular application data, which can reduce the service delay and improve the Quality of Service (QoS) of MUs [9]. Service caching provides a feasible way for constructing a MEC system with low latency and high throughput [10], which is consistent with the development requirements of the fifth-generation (5G) and beyond wireless networks [11]. MEC servers shall cache reasonable application data for different types of CSs to enable fast service execution. It is crucial to implement dynamic service caching for the MEC system to adapt to real-time service requests [12].

MEC systems with service caching offer great advantages in executing the computation intensive and time-critical service requests of MUs. However, the MEC also has many constraints for executing the different types of CSs due to limited computing and storage resources [13]. Specifically, MEC can only cache a limited amount of application data and serve a limited number of MUs in the coverage area [14]. And a potential bottleneck exists in MEC intelligence networks, where the geographically distributed MEC servers may not be able to cooperate to handle the different types of CSs [15]. To tackle this issue, geographical load balancing (GLB) has been considered a promising technique to effectively balance the computing resources among cooperative MEC servers and improve the service performance [16].

The joint optimization of multilayer, multientity, and multidimensional heterogeneous resources of the remote cloud, MEC servers, and MUs' devices is an NP-hard problem, facing the dimensional curse [17]. First, the load balancing in a MEC system with multiple cooperative MEC servers and different types of CSs has become a challenging problem, especially for time-critical CSs. Second, since the service

Manuscript received 25 April 2023; revised 29 August 2023 and 24 October 2023; accepted 8 November 2023. Date of publication 16 November 2023; date of current version 26 March 2024. This work was supported in part by the National Natural Science Foundation of China under Grant 62301307 and Grant 62072096; in part by the International S&T Cooperation Program of Shanghai Science and Technology Commission under Grant 20220713000; in part by the Program for Professor of Special Appointment (Eastern Scholar) at Shanghai Institutions of Higher Learning; in part by the "Shuguang Program" of Shanghai Education Development Foundation and Shanghai Municipal Education Commission; and in part by the Young Top-Notch Talent Program in Shanghai. *(Corresponding author: Wenqian Zhang.)*

Wenqian Zhang is with the College of Information Engineering, Shanghai Maritime University, Shanghai 201306, China (e-mail: zhangwq@shmtu.edu.cn).

Guanglin Zhang is with the College of Information Science and Technology, Donghua University, Shanghai 201620, China (e-mail: glzhang@dhu.edu.cn).

Shiwen Mao is with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL 36849 USA (e-mail: smao@ieee.org).

Digital Object Identifier 10.1109/JIOT.2023.3333826

caching and load balancing decisions are tightly coupled, the main challenge is to jointly optimize the service caching and load balancing [18]. Last but not least, the uncoordinated resource allocation among remote clouds, MEC servers, and MUs' devices leads to a serious waste of network resources. Thus, how to jointly optimize service caching and load balancing for cooperative MEC system to maximize the utilization of system resources is a relevant problem.

Motivated by these challenges, we investigate a hierarchical caching and resources allocation architecture based on deep reinforcement learning (DRL) for the MEC system with multiple cooperative MEC servers. In the service area of each MEC server (i.e., the lower layer), we focus on service caching for different types of CSs and workload offloading for workload transmission in the uplink among MEC servers and MUs. In the global cooperative MEC system (i.e., the upper layer), we consider computation resource allocation and optimize the load balancing among MEC servers. Note that the optimization of the lower layer and the upper layer are tightly coupled. We consider the service caching and resource allocation cost minimization problem over a long-term time horizon given the delay constraints associated with each type of CS. The main contributions made in this article are summarized as follows.

1) We consider service caching, workload offloading, and load balancing in the MEC system with multiple cooperative MEC servers and different types of CSs. We formulate an infinite-horizon cost minimization Markov decision process (MDP). Our goal is to achieve high utilization of system resources and serve more workloads on the premise of meeting the requirements of CSs.
2) We decompose the joint caching and resources allocation optimization problem into two subproblems: a) a lower layer policy optimization and b) an upper layer policy optimization, and design a hierarchical reinforcement learning (HRL)-based caching and resources allocation algorithm (HRL-CRAA) to obtain the optimal policy. To address the coupling of lower layer policy and upper layer policy, we first obtain the caching and offloading decisions from the lower layer optimization and then obtain the load balancing decisions from the upper layer optimization.
3) The feasibility and effectiveness of our proposed HRL-CRAA optimization policy are rigorously analyzed with extensive simulations. The results show that the proposed approach is effective for achieving near optimality in small network sizes and improving the load balancing degree among MEC servers compared with the three baselines: a) HRL-no load balancing (NLB) scheme; b) myopic optimization scheme; and c) randomized offloading scheme.

We organize the remainder of this article as follows. We review related work in Section II and describe the system model in Section III. In Sections IV and V, we present the problem formulation and the proposed DRL framework, respectively. In Section VI, we develop an HRL-based joint caching and resource allocation policy. In Section VII, we present the performance evaluation results and discussions. Finally, we conclude this article in Section VIII.

II. Related Work

With the rapid growth of computation-intensive and time-critical service requests, mobile devices with limited computing and storage resources are facing great challenges. MEC servers can be deployed to enable a variety of applications by offering service execution of different types of CSs. Due to limited MEC resources, only some MU requests from a MEC server's service area can be executed at the MEC server. To make full use of resources and improve the service capabilities of the MEC system, resource allocation and load balancing among cooperative MEC servers have become the focus of many recent works.

MEC was introduced to provide computation and storage capabilities at the edge of the network to improve the QoS of MUs [19], especially for applications with stringent latency requirements [20]. Execution delay is one of the important metrics of MEC design [21], Sun et al. [22] studied the computation task offloading problem in a vehicular edge computing network to reduce the offloading delay. You et al. [23] proposed an energy-efficient resource allocation policy under the execution delay constraint. Yu et al. designed a two-timescale DRL approach for real-time and low-cost computation offloading in [24]. In [25] and [26], offloading and computing resource management schemes were developed to support Internet of Things devices with energy harvesting capability.

MEC brings about several benefits compared to traditional remote cloud-based computing [27]. In addition, MEC can cache popular application data in advance to further reduce the execution delay. Service requests from MUs are usually diverse and change dynamically over time, leading to heterogeneity in both required resources and the popularity of content [28]. To make better use of MEC's limited resources and execute more workloads, collaboration among MEC servers was recently studied in [29], and the load balancing among cooperative MEC servers was studied in [16] and [30]. Wu et al. [16] proposed a Lyapunov optimization-based scheme to investigate the GLB for minimizing the cost of the MEC network. Chen et al. [30] studied computation load balancing to optimize the long-term system performance of the MEC network.

The DRL technology and EI technology have attracted more and more attention to address the problem of network resource allocation [31], [32], [33]. For example, Shen et al. [34] proposed to leverage deep learning-based techniques to predict users' behavior to improve the system's performance. Zhou et al. [35] designed a deep risk-sensitive reinforcement learning policy to minimize the computation and offloading delay. Qian et al. [10] applied DRL to learn MUs' habits and content popularity to optimize the caching and pushing policy. Wang et al. [36] investigated edge caching optimization in a long-term-based federated DRL. Different from these aforementioned works, we consider developing a DRL algorithm to learn the service caching and load balancing features to

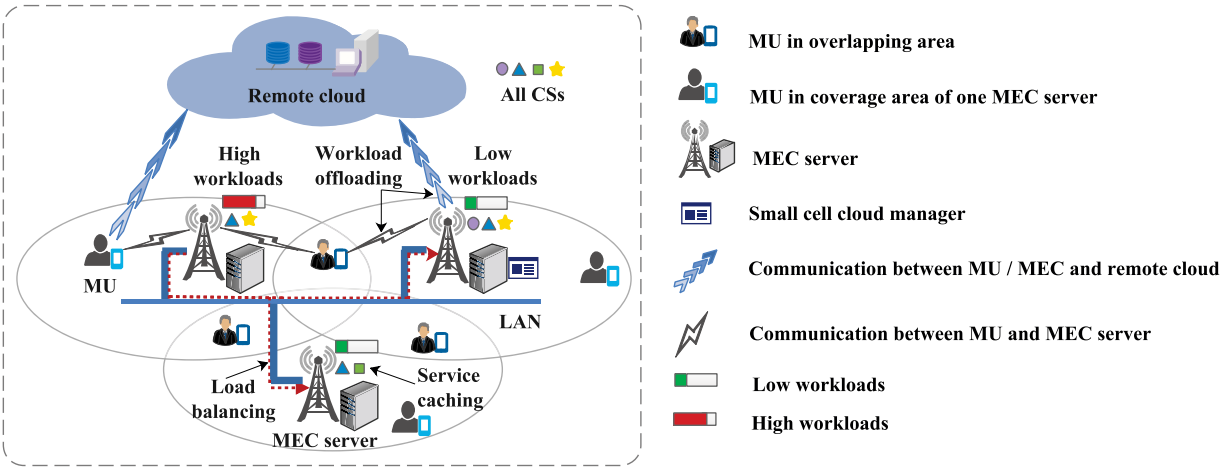

Fig. 1. Architecture of a MEC system with multiple cooperative MEC servers serving MUs.

maximize the utilization of system resources and avoid invalid transmission of workloads.

Through the related work on MEC, we find that only considering the computation delay and energy consumption in MEC networks to schedule workload and allocate resources will cause a waste of network resources. To solve this problem, this work focuses on the joint workload offloading, service caching, and full utilization of network resources optimization. Different from most existing work, optimal scheduling is realized without knowing the energy consumption and delay costs in the process of workload execution. Thus, this work did not pay much attention to consider the process of workload execution in detail in the system model construction.

III. System Model

A. Cooperative MEC System

As illustrated in Fig. 1, we consider a cooperative MEC system consisting of one remote cloud, a set of N MEC servers $\mathcal{N}$, and a set of U MUs $\mathcal{U}$. We denote $\mathcal{U}_n$ as a set of MUs located in the coverage area of MEC server n, where $\mathcal{U}_n \subseteq \mathcal{U}$. The MEC system operates over an infinite period, which is divided into discretized time denoted by $\mathcal{T} \triangleq \{0, 1, \ldots, \}$ and each time slot $t \in \mathcal{T}$, the duration of each time slot t is denoted as τ. The MEC system offers a library $\mathcal{K}$ of K types of CSs to MUs, indicated as $\mathcal{K} = \{1, 2, \ldots, K\}$. Without loss of generality, we assume that the MU u ($u \in \mathcal{U}$) generates one service request for one of the K types of CSs in each time slot. The remote cloud has a high service capability and can execute all types of CSs, while the MEC server n has a finite service capability and can only process one or several types of CSs for MUs (in $\mathcal{U}_n$) located in its coverage area. We assume that the MUs are subscribers of the MEC operator, which are distributed arbitrarily over the coverage areas of the MEC servers and the coverage areas of the MEC servers could overlap. We denote $\mathcal{N}_u \subseteq \mathcal{N}$ as the set of MEC servers that cover MU u.

The MEC servers have finite storage, communication, and computation capabilities, which can precache the applications data related to one or several types of CSs for MUs and provide CSs for MUs located in its coverage area. The MEC server n has a storage capacity C_n, an overall uplink offloading bandwidth W_n for the data transmission among MEC server n and MUs (in $\mathcal{U}_n$) it serves, and a maximum service computation capability F_n (in CPU cycles). In addition, we consider that the MEC system consists of multiple cooperative MEC servers. The MEC servers that have cached the application data related to CS k can exchange their workloads of CS k. The MEC server n with high workloads can switch some workloads to its neighbor MEC servers that have low workloads (which is termed *load balancing* in this article). We denote $\mathcal{M}_n \subseteq \mathcal{N}$ as the set of neighboring MEC servers of MEC server n.

In an effort to enable the joint caching and resource allocation optimization and load balancing among cooperative MEC servers, we introduce a small cell cloud manager (SCM) in the cooperative MEC system, which is a computation control entity. And we deploy the SCM on a MEC server [23], which has the function of collecting global information of the system state and making optimal decisions for MEC servers and MUs at each time slot. Specifically, in each time slot t, the SCM first collects global information on the system state (service requirements of each type of CS, service capability, and service caching state of each MEC server) at the beginning of each time slot. Then, the SCM makes the service caching and workload offloading decisions, and obtaining the optimal occupied storage space and bandwidth allocation for each MU. Next, the SCM derives the optimal load balancing decisions based on the known service caching and workload offloading decisions and computes the optimal computational capability allocation for each type of CS. Finally, the optimal decisions are delivered to MEC servers and MUs. Moreover, the communication traffic among MUs and MEC servers (e.g., service requests and results downloading) are through the wireless channel, while the load balancing among MEC servers is through a wireline local area network (LAN). Note that the transmission scheme among the MEC server and MUs it served is based on orthogonal frequency-division multiple access (OFDMA), and the allocated bandwidth resources of each MEC server are orthogonal and without interference with each other at time slot t [37].

B. Computing Services Request Model

In each time slot, each MU submits its CS request to MEC servers that it can reach. We denote the request of the MU u in the coverage area of MEC server n as $Q_{u,n}(t) = k \in \bar{\mathcal{K}}$, $u \in \mathcal{U}_n$, $n \in \mathcal{N}$, in which $\bar{\mathcal{K}} = \mathcal{K} \cup \{0\}$. If no requests for CS k is submitted to MEC server n at time t, we have $Q_{u,n}(t) = 0$. Assume the request of each MU follows a Markov chain model [38], i.e., $Q_{u,n}(t+1) = \rho_{i,j}^{u,n} \cdot Q_{u,n}(t)$, where $\rho_{i,j}^{u,n}$ is the transition probability from i type of workload to j type of workload for the MU u in the coverage area of MEC server n. In addition, each type of CS has different service requirements related to storage, uplink bandwidth, and execution delay. The required demand of the k type of workload is denoted as $I_{u,n}^k(t) \triangleq \{A_{u,n}^k(t), c_k, w_k, D_k^{\max}\}$, where $A_{u,n}^k(t)$ is the data size of k type of workload generated by MU u in the coverage area of MEC server n at time slot t, and c_k is the occupied storage space of CS k. w_k indicates the required uplink offloading bandwidth of CS k, and $D_k^{\max}$ be the maximum deadline of execution delay of CS k.

C. Service Model

The workloads of CS k within the coverage area of MEC server n can be offloaded to MEC server n where the application data of CS k is cached, and which has sufficient computational capability and bandwidth. In addition, a stressed MEC server n with high workloads can delegate part of its workloads to other MEC servers with low workloads in $\mathcal{M}_n$, where the application data of CS k is cached. If MEC server n and its neighboring MEC servers cannot serve the workloads, the workloads will be further offloaded from MEC server n to the remote cloud to be processed there. We assume that each type of CS is delay sensitive (i.e., the maximum deadline of execution delay of each CS is less than the length of each time slot t), which must be served by a MEC server through the shared LAN or the remote cloud before the end of each time slot [10]. To efficiently utilize the limited communication and computation resources, and to improve the service capability of the cooperative MEC system, the SCM needs to make optimal service caching, workload offloading, and load balancing decisions for each MEC server.

1) Service Caching: At the beginning of each time slot, the MEC server n receives requests for the CS k from the MUs in its coverage area. Then, the MEC server n confirms the service caching state of the applications data associated with the CS k. We use $\mathcal{X}_n(t) = \{x_n^k(t) \in \{0,1\} | k \in \mathcal{K}\}$ to indicate whether the application data associated with the CS k is cached at MEC server n (when $x_n^k(t) = 1$) or not (when $x_n^k(t) = 0$) at time slot t. In addition, the service caching decisions are subject to the MEC server's storage space constraint, given by

$$\sum_{k \in \mathcal{K}} c_k x_n^k(t) \le C_n \quad \forall t \quad \forall n \in \mathcal{N}. \tag{1}$$

Depending on the service capability and the current service caching state of the MEC server, the type of CS request, the SCM makes service caching update decisions for each MEC server. We define $\Delta\mathcal{X}_n(t) = \{\Delta x_n^k(t) \in \{-1, 0, 1\} | k \in \mathcal{K}\}$ as the set for service caching update decisions of the MEC server n related to CS k at time slot t, where -1 means deleting the applications data, 0 means maintaining and 1 means inserting applications data of the CS k. Note that the new service caching state of MEC server n for the CS k at time slot t must belongs to $\{0, 1\}$, i.e., 1) if $x_n^k(t) = 0$, and $\Delta x_n^k(t) \neq -1$ and 2) if $x_n^k(t) = 1$, and $\Delta x_n^k(t) \neq 1$, which needs to satisfy the constraint

$$x_n^k(t) + \Delta x_n^k(t) \in \{0, 1\}. \tag{2}$$

Furthermore, the service caching states of all types of CSs must satisfy the MEC server's storage space constraint at each time slot t. Then, we have the following constraint:

$$\sum_{k \in \mathcal{K}} c_k \left(x_n^k(t) + \Delta x_n^k(t) \right) \le C_n \quad \forall t \quad \forall n \in \mathcal{N}. \tag{3}$$

2) Workload Offloading: According to the service caching decisions of each MEC server related to CS k, the SCM can make workload offloading decisions for workloads of the CS k generated by MUs in the coverage area of each MEC server. We use $\mathcal{Y}_n^k(t) = \{y_{u,n}^k(t) \in \{0,1\} | n \in \mathcal{N}_u \cup \{l\}, u \in \mathcal{U}\}$ to indicate whether the workloads of the CS k generated by MU u (in $\mathcal{U}_n$) are offloaded to MEC server n (when $y_{u,n}^k(t) = 1$) or not (when $y_{u,n}^k(t) = 0$) at time slot t. And the workload offloading decisions for the MEC server n at the time slot t are denoted as $\mathcal{Y}_n(t) = \{\mathcal{Y}_n^k(t) | k \in \mathcal{K}\}$. Similarly, we denote $y_{u,l}^k(t)$ as the offloading decision for offloading workloads to the remote cloud. This happens if the workloads of the CS k generated by the MU u (in $\mathcal{U}_n$) cannot be served by the MEC server n and its neighboring MEC servers (in $\mathcal{M}_n$), then the workloads will be further offloaded to the remote cloud by the MEC server n. Note that the workloads of the CS k generated by the MU u need to be offloaded to one of the MEC servers in $\mathcal{N}_u$, or to the remote cloud. Then, we have

$$\sum_{n \in \mathcal{N}_u \cup \{l\}} y_{u,n}^k(t) = 1 \quad \forall t \quad \forall k \in \mathcal{K}. \tag{4}$$

For the workloads of CS k of the MU u to be offloaded to a MEC server n, as in [39], the service caching decisions and workload offloading decisions need to satisfy

$$y_{u,n}^k(t) \le x_n^k(t) + \Delta x_n^k(t) \quad \forall n \in \mathcal{N}, \ k \in \mathcal{K}. \tag{5}$$

The communication traffic transmission of workloads offloading and results downloading among MEC server n and MUs it served is based on the OFDMA. The MEC server n will allocate the bandwidth resource W_n into S nonoverlapping subcarriers for communication traffic transmission among MEC server n and MUs, it served in time slot t, which is denoted as $\mathcal{S} \triangleq \{0, 1, s, \ldots, S\}$. And we denote $\sum_{k \in \mathcal{K}} w_{k,s} y_{u,n}^k(t)$ as the allocated bandwidth resource of the sth subcarrier for the data transmission between the MEC server n and the MU u in time slot t, which is related to the required uplink offloading bandwidth of each type of CS and the workload offloading decisions of the MU u for different type of CS. $w_{k,s}$ is defined as the uplink offloading bandwidth required by the CS to transmit data over the sth subcarrier. Thus, the sum of occupied offloading bandwidth of all types of CSs generated by MUs

in the coverage area of the MEC server n should not exceed the MEC server's bandwidth capacity, i.e.,

$$\sum_{u\in\mathcal{U}_n}\sum_{k\in\mathcal{K}} w_{k,s}y^k_{u,n}(t) \le W_n \quad \forall t \quad \forall n\in\mathcal{N}. \tag{6}$$

We model the CS k to workloads to be transmitted to the MEC server n as $A^k_n(t) = y^k_{u,n}(t)\mathbf{1}\{n \in \mathcal{N}_u\}(\sum_{u\in\mathcal{U}_n} A^k_{u,n}(t))/(|\eta^k_{u,n}(t)|)$ [30], which is the workloads input of the MEC server n at the time slot t, where $|\eta^k_{u,n}(t)|$ is the number of MEC servers (in $\mathcal{N}_u$) that have cached the application data of the CS k in time slot t.

3) Load Balancing: To efficiently utilize the limited communication of MEC servers, we focus on load balancing among MEC servers and ignore the workloads execution by MUs in this work. In each time slot t, the workloads of the CS k offloaded to the MEC server n will be processed at the server locally, or further delegated to one of the neighboring MEC servers in $\mathcal{M}_n$. Let $\boldsymbol{\beta}^k_{n\cdot}(t) = \{\beta^k_{nm}(t)\}_{m\in\mathcal{N}}$ denote the load balancing decisions for the CS k of MEC server n in time slot t, where $\beta^k_{nn}(t)$ means that the workloads are retained and processed at the MEC server n, and $\beta^k_{nm}(t)$ indicates the delegated workloads from the MEC server n to the MEC server m ($m \in \mathcal{M}_n$). Therefore, we obtain the workload output of the MEC server n at time slot t as $\sum_{m\in\mathcal{N}}\beta^k_{nm}(t) = [\sum_{m\in\mathcal{M}_n}\beta^k_{nm}(t)] + \beta^k_{nn}(t)$. Denote $\boldsymbol{\beta}_n(t) = \{\boldsymbol{\beta}^k_{n\cdot}(t)\}_{k\in\mathcal{K}}$ as the set of overall load balancing decisions of the MEC server n for all types of CSs, and $\boldsymbol{\beta}(t) = \{\boldsymbol{\beta}_n(t)\}_{n\in\mathcal{N}}$ is the set of overall load balancing decisions in our MEC system.

Furthermore, denote $\boldsymbol{\beta}^k_{\cdot n}(t) = \{\beta^k_{mn}(t)\}_{m\in\mathcal{M}_n}$ as the workload that the MEC server n receives from its neighboring MEC servers in $\mathcal{M}_n$. The total workloads of the CS k processed by the MEC server n at time slot t are given by $\alpha^k_n(t) \triangleq [\sum_{m\in\mathcal{M}_n}\beta^k_{mn}(t)] + \beta^k_{nn}(t)$. In addition, the load balancing decisions need to satisfy the following constraints.

1) The workloads for load balancing among MEC server n and its neighboring MEC servers must be nonnegative, i.e., $\beta^k_{nm}(t) \ge 0$, for all k, n, m_n, and t.
2) To avoid transmission loops among MEC servers, the workloads input for the CS k to MEC server n should be equal to its output [30], i.e.,

$$A^k_n(t) = \sum_{m\in\mathcal{N}}\beta^k_{nm}(t) \quad \forall n \quad \forall k \quad \forall t. \tag{7}$$

3) The workloads processed by the MEC server n cannot exceed its computational capability, i.e., the stable condition of the M/M/1 queueing model [40], which is given by

$$\begin{cases} \alpha^k_n(t) = f^k_n - 1/D^k_{\max} \\ \sum_{k\in\mathcal{K}} f^k_n \le F_n \quad \forall t \quad \forall n\in\mathcal{N} \end{cases} \tag{8}$$

where f^k_n is the computational resources allocated by MEC server n for processing CS k workloads.

The main parameters used in this article are summarized in Table I.

TABLE I
NOTATION

Symbol	*Definition*
N	The number of MEC servers
U	The number of mobile users in cooperative MEC system
K	The total number of types of CSs
$\mathcal{N}$	Index set of MEC servers
$\mathcal{N}_u$	Index set of the MEC servers that covering MU u
$\mathcal{U}$	Index set of mobile users
$\mathcal{K}$	Index set of computing services
$\mathcal{T}$	Index set of time slots
$\mathcal{M}_n$	Index set of the neighboring MEC servers of MEC server n
C_n	The fixed storage capacity of MEC servers n
W_n	The overall uplink offloading bandwidth among MEC server n and MUs served by MEC server n
F_n	The maximum service computation capability (in CPU cycles) of MEC server n
c_k	The occupied storage space of CS k
w_k	The required uplink offloading bandwidth of CS k
$D^k_{\max}$	The maximum deadline of execution delay of CS k
$A_{n,k}(t)$	The workloads for CS k generated by MUs in coverage area of MEC server n in time slot t
$x_{n,k}(t)$	The service caching decision of MEC server n for CS k in time slot t
$\Delta x_{n,k}(t)$	The service caching update decision of MEC server n for CS k in time slot t
$y_{n,k}(t)$	The workload offloading decision of MEC server n for CS k in time slot t
$A^k_n(t)$	The workloads of CS k to transmitted to MEC server n
$\|\eta_k(t)\|$	The number of MEC servers that cached applications data associated with CS k in time slot t
$\boldsymbol{\beta}^k_{n\cdot}(t)$	The load balancing decisions for CS k of MEC server n at time slot t
$\alpha^k_n(t)$	The overall workloads of CS k processed by MEC server n at time slot t
f^k_n	The computation resource allocated by MEC server n for CS k

IV. Problem Formulation

A. Communication Resource Allocation

From the system model described in the previous section, the service caching and workload offloading decisions have a big influence on the communication resource allocation of the MEC system. Similar to [10], in an effort to maximize the communication resource utilization, we introduce a cost function $\phi(\cdot)$ to indicate the fluctuation of the communication resource, which is a convex function, and "$\cdot$" represents the related fluctuation parameter. Specifically, let $\phi(C) = C^2$ be the cost of storage usage, where C is the state of storage space for service data caching at time slot t, i.e., C represents the remaining storage space in the system at time slot t. Similarly, denote $\phi(W) = W^2$ as the cost of bandwidth usage, where W is the state of uplink offloading bandwidth for data transmission, i.e., W represents the remaining bandwidth resource in the system at time slot t.

Without loss of generality, the service caching and workload offloading decisions affect the fluctuation of storage space, and change of uplink offloading bandwidth usages, respectively. Based on the definition of the $\phi(C) = C^2$ and $\phi(W) = W^2$, we analyze that when the minimum value of the function is

obtained, the system residual resource is the least. Thus, to make full use of the communication resource (space storage and bandwidth), we must obtain the minimum values of the cost functions. Furthermore, based on the cost function $\phi(\cdot)$, when the fluctuation parameter is small, the value of $\phi(\cdot)$ is going to be small. In other words, a small fluctuation of the parameter results in a high utilization of communication resources. Moreover, we define the average sum cost of service caching and workload offloading of the MEC server n at time slot t as $\phi(CX_n(t)) + \phi(WY_n(t))$, where $\phi(CX_n(t)) = [C_n - \sum_{k\in\mathcal{K}} c_k x_n^k(t)]^2$ and $\phi(WY_n(t)) = [W_n - \sum_{k\in\mathcal{K}} w_k y_{u,n}^k(t)]^2$.

B. Computation Resource Allocation

Each MEC server will process different types of workloads of CSs. Load balancing among cooperative MEC servers will help to better utilize MEC system's computational resources and mitigate the congestion of workload processing. We use the variance $\text{var}(\alpha_n^k(t))$ of the overall workloads of the CS k processed by the MEC server n at time slot t to represent the workload differences and performance of the service capabilities among MEC servers [13], which is given by

$$\text{var}\left(\alpha_n^k(t)\right) = \frac{1}{|N|}\sum_{n\in\mathcal{N}}\left(\alpha_n^k(t) - \frac{1}{|N|}\sum_{n\in\mathcal{N}}\alpha_n^k(t)\right)^2. \quad (9)$$

For load balancing among MEC servers, the key is to minimize the variance $\text{var}(\alpha_n^k(t))$ of computation resource allocation concerning each type of CS.

C. Problem Formulation

To efficiently utilize the communication and computation resources of the MEC system, the MEC system operator needs to make optimal service caching, workload offloading, and load balancing decisions in each time slot. Denote the control vectors set of our cooperative MEC system for the MEC server n at time slot t as a policy $\boldsymbol{\pi}(t) \triangleq \{\Delta\mathcal{X}_n(t), \mathcal{Y}_n(t), \boldsymbol{\beta}_n(t)\}$. We aim to find the optimal service caching update decisions $\Delta\mathcal{X}_n(t) = \{\Delta x_n^1(t), \Delta x_n^2(t), \ldots, \Delta x_n^K(t)\}$, workload offloading decisions $\mathcal{Y}_n(t) = \{\mathcal{Y}_{1,n}^k(t), \mathcal{Y}_{2,n}^k(t), \ldots, \mathcal{Y}_{U_n,n}^k(t)\}$, and load balancing decisions $\boldsymbol{\beta}_n(t) = \{\boldsymbol{\beta}_{n\cdot}^1(t), \boldsymbol{\beta}_{n\cdot}^2(t), \ldots, \boldsymbol{\beta}_{n\cdot}^K(t)\}$.

We formulate the joint caching and resource allocation problem as a cost minimization optimization problem over a long-term time horizon. The objective function is defined as the weighted, time-averaged sum of service caching and workload offloading costs, as well as the balancing degree of computational resources allocation among MEC servers, which is given by

$$\varphi(\boldsymbol{\pi}(t)) = \lim_{T\to\infty}\frac{1}{T}\sum_{t=0}^{T-1}\sum_{n=1}^{N}\mathbb{E}\Big\{\xi_1\phi(CX_n(t)) + \xi_2\phi(WY_n(t)) + \xi_3\cdot b^n\Big\} \quad (10)$$

where $\mathbb{E}\{\cdot\}$ is the expectation over MEC servers, and ξ_λ, $\lambda \in \{1,2,3\}$, represent the weights of parameters in the objective function. Similar to the variance of computational resource allocation for each type of CS on each MEC server in (9), we define $b^n = (1/|N|)\sum_{n\in\mathcal{N}}([\sum_{k\in\mathcal{K}}\alpha_n^k(t)] - (1/|N|)\sum_{n\in\mathcal{N}}\sum_{k\in\mathcal{K}}\alpha_n^k(t))^2$ as the load balancing degree among MEC servers on the MEC server n for all types of CS.

The optimization problem is formulated as

$$\varphi^* = \min_{\boldsymbol{\pi}(t)}\varphi(\boldsymbol{\pi}(t)) \quad \text{s.t.} \quad (1)\sim(8) \quad (11)$$

where φ^* is the optimal averaged system cost in the long-term time horizon; (1) and (2) are the storage space and service caching state constraints of each MEC server n, respectively. Constraint (3) represents the storage space constraint of MEC server n after service caching state update. Constraint (4) ensures that the workloads of the CS k in the coverage area of MEC server n should be offloaded to exactly a connected MEC server or the remote cloud. Constraint (5) describes the relationship between service caching and workload offloading decisions. Constraint (6) enforces the limit of MEC server uplink offloading bandwidth capacity. Constraint (7) ensures that the workload input and output of the MEC server n are equal. Constraint (8) indicates the workloads executed by the MEC server n cannot exceed its service capability.

The formulated joint caching and resource allocation problem is considered as an infinite-horizon cost minimization MDP. Without knowledge of the transition probabilities of MU's CSs requests, it is a challenging problem to solve. Fortunately, based on the centralized framework, the SCM can make accurate instructions after grasping global information with fast calculation speed. By using SCM, the load balancing among multiple MEC servers can be better implemented and resource allocation can be managed. However, the computational complexity of the centralized framework will increase with the number of MEC servers and the number of MUs they serve. Since the workload offloading decisions are based on service caching decisions, and load balancing decisions influence the workload offloading decisions, we will develop an HRL-based caching and resource allocation policy in Section VI to deal with the coupling of three decisions. Therefore, we propose to develop a reinforcement learning-based algorithm to learn the MU's request models and derive the optimal policy π^* [10]. We will define the three key elements (i.e., state, action, and reward) of the proposed reinforcement learning algorithm in the next section.

V. Deep Reinforcement Learning Framework

A. State

With the proposed algorithm, the states consist of the CS request state and service caching state of the MEC servers at each time slot. The request state of CS k contains the workloads, the service requirements, the MUs' locations that generate the workloads, and the corresponding servers $\mathcal{N}_u$ of the MUs. For the convenience of analysis, we use the MEC server n as an example below. Let $\mathcal{A}_n(t) = \{A_n^k(t)|k\in\mathcal{K}\}$ be the CS request state, and $\mathcal{X}_n(t) = \{x_n^k(t)|k\in\mathcal{K}\}$ the service caching state. Furthermore, we denote $\mathcal{S}(t) = \{\mathcal{A}_n(t), \mathcal{X}_n(t)\}$ as the current state of MEC server n. Since $A_n^k(t) \in \bar{\mathcal{A}}_n^k$ and $|\bar{\mathcal{A}}_n^k| = A_n^k + 1$, we obtain the dimension of the CS request state as $|\mathcal{A}_n(t)| = (A_n^k+1)^K$. Moreover, since $x_n^k(t) \in \{0,1\}$, we

obtain the dimension of the service caching state as $|\mathcal{X}_n| = 2^K$. Therefore, the dimension of the entire system state of MEC server n is $|\mathcal{S}| = (A_n^k + 1)^K \times 2^K$.

B. Action

In the cooperative MEC system, we define three types of actions, including 1) service caching update; 2) workload offloading; and 3) load balancing. We denote $\mathcal{A}(t) = \{\boldsymbol{\beta}_n(t), \Delta\mathcal{X}_n(t), \mathcal{Y}_n(t)\}$ as the action of MEC server n at time slot t. We use $\boldsymbol{\beta}_n(t) = \{\boldsymbol{\beta}_{n\cdot}^k(t)|k \in \mathcal{K}\}$ to denote the load balancing action for the MEC server n, where $\boldsymbol{\beta}_{n\cdot}^k(t) = \{\beta_{n\cdot}^{k,1}, \ldots, \beta_{n\cdot}^{k,i}, \ldots, \beta_{n\cdot}^{k,I-1}\}$, and $\beta_{n\cdot}^{k,i} = 1$ represents that the data size for load balancing of the ith discretized grade according to the available maximization of workloads transmission $\beta_{n\cdot}^{k,\max}$ between MEC server n and its neighboring MEC servers [37]. Since the workload input of the k type of CS to MEC server n is equal to its output, we set $|A_n^k(t)|$ as the maximum workloads for load balancing at time slot t. According to $A_n^k(t) \in \bar{\mathcal{A}}_n^k$ and $|\bar{\mathcal{A}}_n^k| = A_n^k + 1$, we derive the load balancing action dimension as $|\beta| = [(1/I)(A_n^k + 1)]^{NK}$. In addition, with the service caching decisions $\Delta x_n^k \in \{-1, 0, 1\}$ and the workload offloading decisions $y_n^k \in \{0, 1\}$, the corresponding dimensions are 3^K and 2^K, respectively. Thus, the dimension of the overall system action space of MEC server n at time slot t is $|\mathcal{A}| = [(1/I)(A_n^k + 1)]^{NK} \times 3^K \times 2^K$.

C. Reward

The reward function needs to reflect the objective of our proposed joint caching and resource allocation optimization problem, i.e., maximizing the communication and computation resources utilization. We consider the objective as the environment feedback in the DRL system. Denote the reward function as follows:

$$r \triangleq -\Big\{\xi_1\phi(CX_n(t)) + \xi_2\phi(WY_n(t)) + \xi_3 \cdot b^n\Big\}. \tag{12}$$

With the formulated reinforcement learning framework and state–action value function (Q value), we obtain the Bellman equation as

$$Q(s(t), a(t)) = \mathbb{E}\Big[r + \gamma Q'\Big(s(t+1), a(t+1)\Big)|s(t), a(t)\Big] \tag{13}$$

where γ is the discount rate that indicates the impact of future reward on current cumulative reward. $s(t) = \{\mathcal{A}_n(t), \mathcal{X}_n(t)\}$ is the current MEC system state, r is defined as reward in (12), and $s(t+1) = \{\mathcal{A}_n(t+1), \mathcal{X}_n(t+1) + \Delta\mathcal{X}_n(t+1)\}$ is the system state in the next time slot. The corresponding optimal policy π^* can be derived as follows:

$$\pi^*(s(t), a(t)) = \underset{\boldsymbol{\beta}_n, \Delta\mathcal{X}_n, \mathcal{Y}_n}{\arg\max}\ Q(s(t), a(t)). \tag{14}$$

VI. HRL-Based Joint Caching and Resources Allocation Policy

As mentioned, due to the large dimensions of the state and action spaces, it is difficult to search the state–action in the large space for the optimal policy. Applying reinforcement learning directly will incur slow convergence. Furthermore, because the workload offloading decisions are based on service caching decisions, and load balancing decisions influence the workload offloading decisions. To address the slow convergence in large space and decisions coupled problem, we decompose the joint caching and resource allocation problem into two subproblems, i.e., Subproblem 1 and Subproblem 2, and then present HRL based on deep Q networks (DQNs) to solve these subproblems.

Substitute (12) into (13), and according to (14), we can decompose the joint caching and resource allocation problem into two subproblems.

1) *Subproblem 1 (Lower Layer Policy Optimization):* For given $\boldsymbol{\beta}$, Subproblem 1 is given by

$$\begin{aligned}\varphi_{\text{Low}}^* = \max_{\Delta\mathcal{X}_n, \mathcal{Y}_n}\ &\Big\{\gamma Q'_{\text{Low}}(s_{\text{Low}}(t+1), a_{\text{Low}}(t+1)) \\ &-\Big[\xi_1\phi(CX_n(t)) + \xi_2\phi(WY_n(t)) + \xi_3 \cdot b^n\Big]\Big\} \\ \text{s.t.}\quad &(1)\sim(6).\end{aligned} \tag{15}$$

2) *Subproblem 2 (Upper Layer Policy Optimization):* For given $\mathcal{X}$, $\mathcal{Y}$, and $\Delta\mathcal{X}$, Subproblem 2 is given by

$$\begin{aligned}\varphi_{Up}^* = \max_{\boldsymbol{\beta}_n}\ &\Big\{\gamma Q'_{Up}\big(s_{Up}(t+1), a_{Up}(t+1)\big) \\ &-\Big[\xi_1\phi(CX_n(t)) + \xi_2\phi(WY_n(t)) + \xi_3 \cdot b^n\Big]\Big\} \\ \text{s.t.}\quad &(8).\end{aligned} \tag{16}$$

Since the decisions of the lower layer optimization affect the upper layer optimization decisions, the two subproblems are coupled. Hence, we first obtain the service caching and workload offloading decisions from Subproblem 1 by using DQN of the first layer (DQN1). Then, we take the known decision of the lower layer into consideration to obtain the load balancing decisions from Subproblem 2 based on the DQN of the second layer (DQN2). We next present an HRL-CRAA for solving the two subproblems.

A. Deep Q Network

DQN combines deep learning with reinforcement learning and uses a deep neural network (DNN) with parameter ω to generate action values and approximate the Q value. Fig. 2 shows the detailed DQN architecture. For given state $s(t)$, action $a(t)$, and reward $r(t)$, we set P as the capacity of the experience memory D, as well as the target network and evaluation network with parameters ω^- and ω, respectively.

In each step t, the input to the evaluation network is the current system state $s(t)$, and the output is the Q value $Q(s(t), a(t); \omega)$ of each action. The action $a(t)$ can be selected based on Q value $Q(s(t), a(t); \omega)$ of the evaluation network with the ϵ-greedy strategy, i.e., choosing a random action $a(t)$ with probability ϵ, or choosing the optimal policy given by $a(t) = \arg\max_{a(t)} Q(s(t), a(t); \omega)$, where the Q value is in the evaluation network.

After choosing the action $a(t)$, we obtain the corresponding reward $r(t)$ and the next state $s(t+1)$. All of these are put into an experience memory as training samples. During the training stage, random batches of data are sampled from the experience memory to train the DNN. After the DNN is well trained, the approximated Q value $Q(s(t), a(t); \omega)$ will approach the target

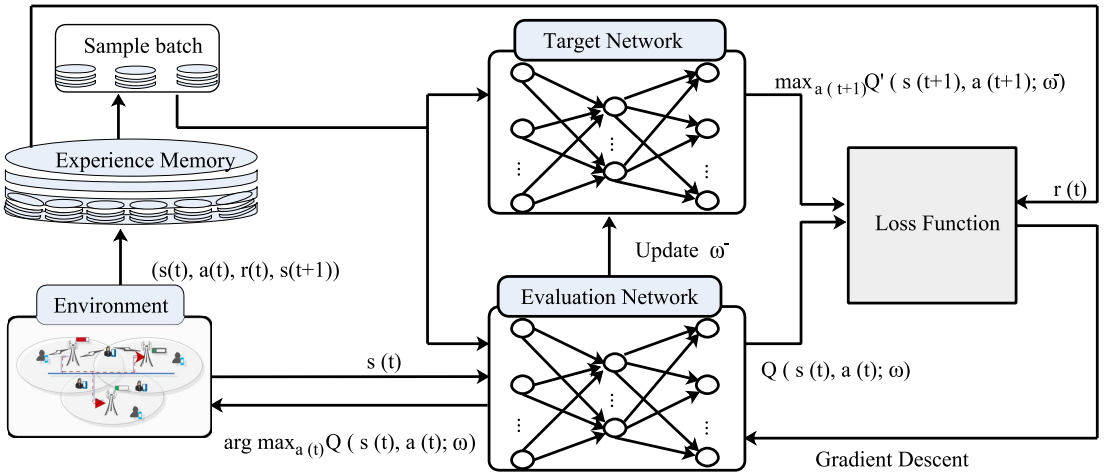

Fig. 2. Framework of the DQN model.

Q value $Q^*(s(t), a(t))$ in the target network. To minimize the discrepancy between the Q values from the evaluation network and the target network, we adopt the following loss function for modeling training:

$$\mathcal{L}_i(\omega) = \mathbb{E}\left[\left(r + \gamma \max_{a(t+1)} Q'(s(t+1), a(t+1); \omega^-) - Q(s(t), a(t); \omega)\right)^2\right] \quad (17)$$

where $r+\gamma \max_{a(t+1)} Q'(s(t+1), a(t+1); \omega^-)$ is the Q value calculated by the target network with parameter ω^- in iteration i. In order to train the DNN, we use the gradient descent algorithm by differentiating the loss function (17) with respect to the weights as follows:

$$\nabla_\omega L_i(\omega) = \mathbb{E}\left[\left(r + \gamma \max_{a(t+1)} Q'(s(t+1), a(t+1); \omega^-) - Q(s(t), a(t); \omega)\right)\nabla_\omega Q(s(t), a(t); \omega)\right]. \quad (18)$$

Note that the initial parameters of the evaluation network and the target network take the same values. The parameters ω of the evaluation network are updated in each step, but the parameters ω^- of the target network are updated for every fixed J steps. We introduce the DQN algorithm to solve Subproblem 1 and Subproblem 2 in Sections VI-B and VI-C, respectively.

B. Service Caching and Workload Offloading Policy

We first examine the optimization of the service caching and workload offloading policy by using DQN for the *lower layer policy optimization* in Subproblem 1. Due to the large dimensions of the state and action spaces, we use MEC server n as an example below to facilitate the analysis. We redefine the state of DQN1 as $s_{\text{Low}}(t) = \{A_n^k(t) | k \in \mathcal{K}\}$. In addition, we obtain the reduced version of actions for DQN1 as $a_{\text{Low}}(t) = \{y_n^k(t) \in \{0, 1\}, \Delta x_n^k(t) \in \{-1, 0, 1\} | k \in \mathcal{K}\}$. Since the given load balancing decisions do not affect the low layer optimization in Subproblem 1, we redefine the reward of the lower layer optimization as $r_{\text{Low}} \triangleq -[\xi_1 \phi(CX_n(t)) + \xi_2 \phi(WY_n(t))]$. Substituting the redefined state, action, and reward into the Bellman equation (13), we derive the corresponding optimal policy π^*_{Low} for Subproblem 1 as follows:

$$\begin{aligned} &\pi^*_{\text{Low}}(s_{\text{Low}}(t), a_{\text{Low}}(t)) \\ &= \arg\max_{\Delta\mathcal{X}_n, \mathcal{Y}_n} \left\{ Q'_{\text{Low}}(s_{\text{Low}}(t+1), a_{\text{Low}}(t+1)) \right. \\ &\quad \left. - \left[\xi_1 \phi(CX_n(t)) + \xi_2 \phi(WY_n(t))\right] \right\} \\ &\text{s.t.} \quad (1) \sim (6). \end{aligned} \quad (19)$$

The procedure is shown in Algorithm 1. Given the three key elements (i.e., state, action, and reward), we first initialize the corresponding parameters D_{Low}, P_{Low}, ω_{Low}, and ω^-_{Low}. According to the process introduced in Section VI-A, in each step t, action $a_{\text{Low}}(t)$ is chosen based on the evaluation network's output with the ϵ-greedy strategy. Then, the obtained new sample $(s_{\text{Low}}(t), a_{\text{Low}}(t), r_{\text{Low}}(t), s_{\text{Low}}(t+1))$ is stored in the experience reply memory D_{Low}. Procedure 1 is executed to update the evaluation network. After the DQN1 is well trained, we execute Procedure 2 to call the trained DQN1 network to obtain service caching and workload offloading decisions. Note that Procedures 1 and 2 are the common procedures used in both DQN1 and DQN2 for updating the evaluation network and calling the trained network, respectively.

C. Load Balancing Policy Among MEC Servers

Next, we examine the optimization of load balancing policy among MEC servers by using the DQN algorithm to solve Subproblem 2. The optimization of the lower layer affects the upper layer's decisions, which leads to a complex and dynamic MEC environment. In the upper layer optimization, the caching and offloading decisions from the lower layer optimization become the state for load balancing optimization. Thus, the state for DQN2 is given by $s_{Up}(t) = \{A_n^k(t), y_n^k(t), \Delta x_n^k(t) | k \in \mathcal{K}\}$, and the action is redefined as $a_{Up}(t) = \{\boldsymbol{\beta}_{n\cdot}^k(t) | k \in \mathcal{K}\}$. In addition, the reward $r_{Up} = r$, which is defined in (12). Similarly, substituting the redefined state, action, and reward into the Bellman equation (13), we

Algorithm 1: Lower Layer Policy Optimization by Using DQN

Require: Discount γ_{Low}, exploration rate ε_{Low}, replay memory capacity P_{Low};
Initialize the replay memory D_{Low} to capacity P_{Low};
Initialize the evaluation network with parameters ω_{Low};
Initialize the target network with parameters ω_{Low}^{-};
for *each episode i* **do**
  Initialize state $s_{Low,1}$;
  for *each step t* **do**
    With probability ε_{Low} choose a random action $a_{Low}(t)$;
    Otherwise select action $a_{Low}(t)$ by evaluation network with parameter ω_{Low} based on (19);
    Execute action $a_{Low}(t)$ in emulator, obtain reward r_{Low} and new state $s_{Low}(t+1)$;
    Store the following transition in D_{Low}: $(s_{Low}(t), a_{Low}(t), r_{Low}(t), s_{Low}(t+1))$;
    Execute Procedure 1 ;
    Each J_{Low} steps reset $\omega_{Low}^{-} = \omega_{Low}$;
  end
end
Execute Procedure 2 to obtain service caching and workloads offloading policies;

Procedure 1: Evaluation Network Updating Procedure

Sample random minibatch of transitions (s_i, a_i, r_i, s_{i+1}) from D ;
if *episode terminates at step* $i+1$ **then**
  Set $y_i = r_i$;
else
  Set $y_i = r_i + \gamma \max_{a_{i+1}} Q'\left(s_{i+1}, a_{i+1}; \omega^{-}\right)$;
end
Execute (17) and (18) to obtain the gradient descent;

Procedure 2: Running Procedure

Require: Trained evaluation network with parameters ω;
Obtain initial state s_1;
for *each step t* **do**
  Select $a(t) = \arg\max_{a(t)} Q(s(t), a(t); \omega)$;
  Execute action $a(t)$ in the emulator;
  Observe reward $r(t)$ and new state $s(t+1)$;
end

derive the corresponding optimal policy π_{Up}^{*} for Subproblem 2 as follows:

$$\begin{aligned}&\pi_{Up}^{*}\left(s_{Up}(t), a_{Up}(t)\right)\\&= \arg\max_{\boldsymbol{\beta}_n} \Big\{ Q'_{Up}\left(s_{Up}(t+1), a_{Up}(t+1)\right)\\&- \left[\xi_1 \phi(CX_n(t)) + \xi_2 \phi(WY_n(t)) + \xi_3 \cdot b^n\right] \Big\}\\&\text{s.t. (8).}\end{aligned} \tag{20}$$

Algorithm 2: HRL-CRAA

Require: Discount γ_{Up}, exploration rate ε_{Up}, replay memory capacity P_{Up};
Initialize the replay memory D_{Up} to capacity P_{Up};
Initialize the evaluation network with parameters ω_{Up};
Initialize the target network with parameters ω_{Up}^{-};
for *each episode i* **do**
  Initialize state $s_{Low,1}$;
  for *each step t* **do**
    Execute Algorithm 1 to obtain the caching and offloading decisions of lower layer optimization;
    Update $s_{Low,1}$ to $s_{up,1}$ with the lower layer policy;
    With probability ε_{Up} choose a random action $a_{Up}(t)$;
    Otherwise choose $a_{Up}(t)$ by evaluation network with parameter ω_{Up} based on (20);
    Execute action $a_{Up}(t)$ in emulator, obtain reward r_{Up} and new state $s_{Up}(t+1)$;
    Store the following transition in D_{Up}: $(s_{Up}(t), a_{Up}(t), r_{Up}(t), s_{Up}(t+1))$;
    Execute Procedure 1;
    Each J_{Up} steps reset $\omega_{Up}^{-} = \omega_{Up}$;
  end
end
Execute Procedure 2 to obtain load balancing policies;

The HRL-CRAA is presented in Algorithm 2. After executing Algorithm 1, we obtain the service caching and workload offloading decisions in the *lower layer policy optimization*. Algorithm 2 is mainly to solve the load balancing decisions in the *upper layer policy optimization* based on DQN2 with the known optimal solution of Algorithm 1. Furthermore, the process of DQN2 is similar to that of DQN1 as given in Algorithm 1. Note that the state s_{Up} is updated by state s_{Low} and the lower layer policies at each step t. We omit the description of the DQN2 process for brevity. After executing Algorithm 2, we obtain the load balancing decisions.

VII. Simulation Results and Discussion

A. Parameter Setting

In this section, we present the simulation study to evaluate the performance of our proposed HRL-CRAA. The operating environment of the experiment is based on a laptop with Intel Core i7-6500U 2.5-GHz processor and 16-GB RAM. We consider $N = 3$ MEC servers and $U = 6$ MUs are uniformly distributed in the coverage areas of the MEC servers as shown in Fig. 1, and the cooperative MEC system is operated on discrete time slots with $\tau = 1$ s. In each time slot, each MU generates one type of CSs request from a library with $K = 2$. We assume that the popularity of each type of CS follows a Zipf distribution and the shape parameter is 0.8 [39]. The workload arrivals of each type of CS request are randomly distributed within [600, 700] Mb [29]. Furthermore, we set the storage capacity of MEC server n, i.e., C_n, randomly within [200, 600] GB [39], and the overall uplink offloading

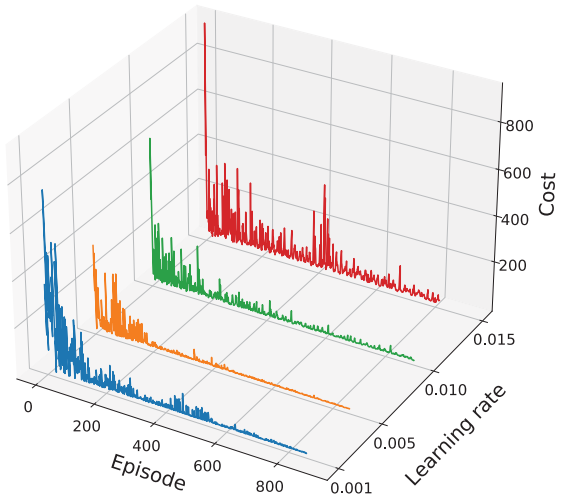

Fig. 3. Convergence performance of HRL-based lower layer optimization under different learning rates.

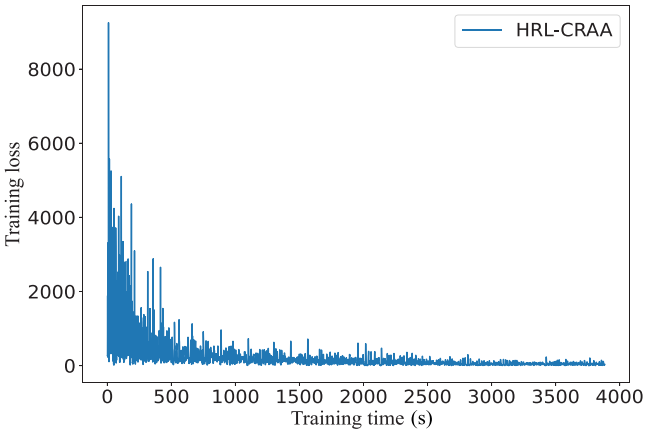

Fig. 4. Convergence performance of HRL-CRAA under a learning rate of 0.01: training loss.

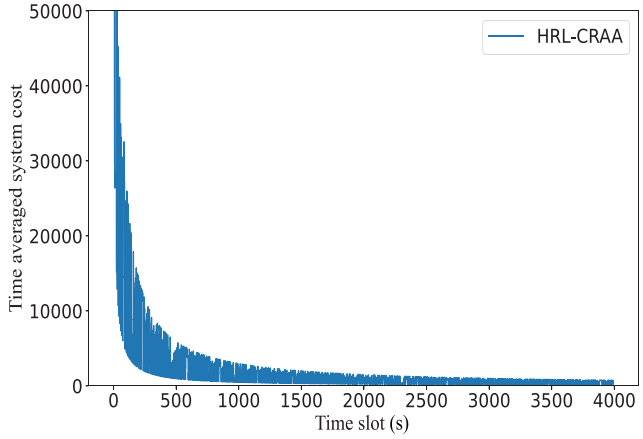

Fig. 5. Convergence performance of HRL-CRAA under a learning rate of 0.01: time averaged system cost.

bandwidth W_n as 25 MHz [1]. The service computational capability F_n is uniformly distributed between 800 and 3000 (in CPU cycles). Moreover, we set the occupied storage space of each CS as $c_k \in [100, 200]$ GB. The required uplink offloading bandwidth is $w_k \in [5, 20]$ MHz, and the maximum deadline of execution delay is randomly set within $D^k_{\max} \in [2, 7] \times 10^{-3}$s [16]. In addition, we set $\xi_1 = \xi_2 = \xi_3 = 1$ as the weight of parameters in the objective function (10). In an effort to ensure the matrix of the training action $a(t)$ of the DQN architecture fits the model environment, we preprocess the generated decision matrix and select the appropriate data as the training action $a(t)$ in simulation based on the mathematical constraint in (2). In this way, we can guarantee the accuracy of the DQN training.

B. Benchmark Schemes Designing

We compare our proposed HRL-CRAA scheme with the following three baseline schemes.

1) HRL-NLB: This is an HRL-based caching and resource allocation policy but with NLB among the MEC servers. That is, the arrived workloads to the MEC server n will be processed by the MEC server. If the MEC server n cannot meet the service requirements, the workloads will be further offloaded to the remote cloud.

2) Myopic Optimization: In this approach, we set the discount $\gamma = 0$ in the DQN training process. The MEC system pays attention to the current reward, and we obtain the actions without considering the temporal correlations between future states and decisions.

3) Randomized Offloading: In this approach, the workload offloading decision is randomly selected from the set of offloading actions with equal probability.

C. Results and Discussion

We first examine the convergence performance of our proposed HRL-CRAA scheme. As shown in Fig. 3, we set the learning rate to 0.001, 0.005, 0.01, and 0.015 in our simulations to present the convergence of the cost. With the increase in learning rate (e.g., from 0.001 to 0.005), the convergence speed of HRL-CRAA is accelerated. A small learning rate leads to a slow convergence speed and a longer training time. And if the learning rate is too large, the algorithm may converge to a poor solution, or it may not converge at all, as the case when learning rate = 0.015 shows. Therefore, we set the learning rate in the range of 0.005 to 0.01 in our simulation experiment.

In Figs. 3 and 4, we present the convergence performance of the proposed scheme concerning cost and training loss. It can be seen that there is a large cost (training loss) at the beginning of the training process because the DRL agent has not learned enough information to make sound caching and resource allocation decisions. As the number of episodes increases, the cost gradually decreases until it reaches a relatively stable value. Fig. 3 shows the convergence for the lower layer optimization, and Fig. 4 shows the convergence for the proposed HRL-CRAA scheme. In addition, we present the convergence performance with respect to time-averaged system cost of HRL-CRAA under a learning rate of 0.01 in Fig. 5. As the training time increases, the reward gradually increases and the system cost gradually decreases as well, which proves the rationality of our reward function setting in (12).

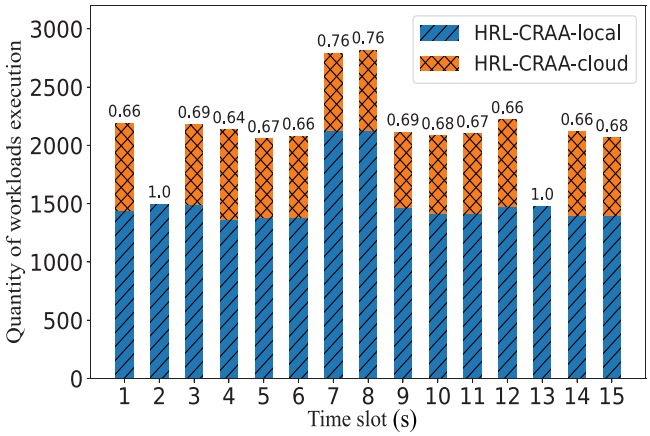

Fig. 6. Resource utilization of HRL-CRAA in different time slots.

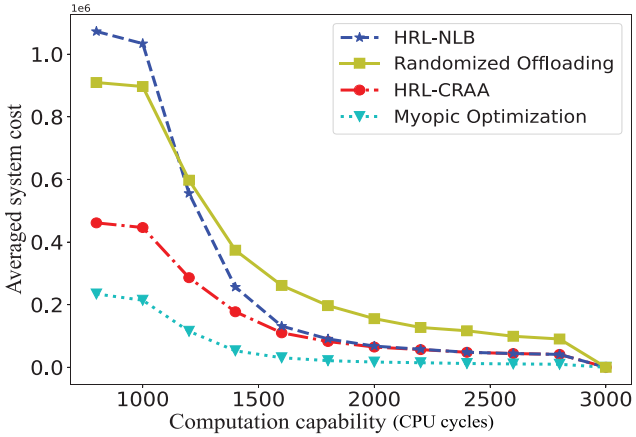

Fig. 8. Averaged system cost with different computational capabilities with different schemes.

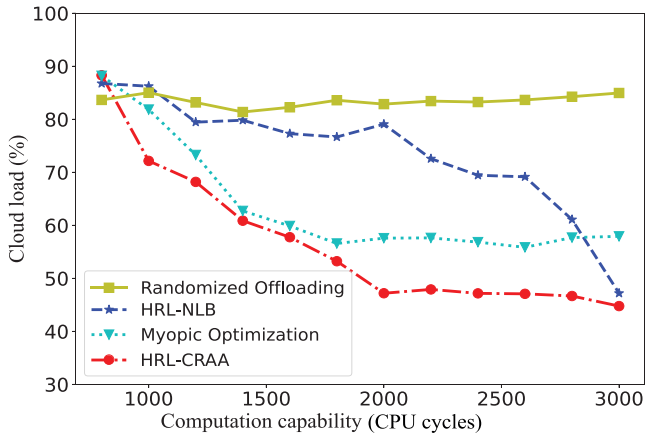

Fig. 7. Cloud load ratio for different computational capabilities with different schemes.

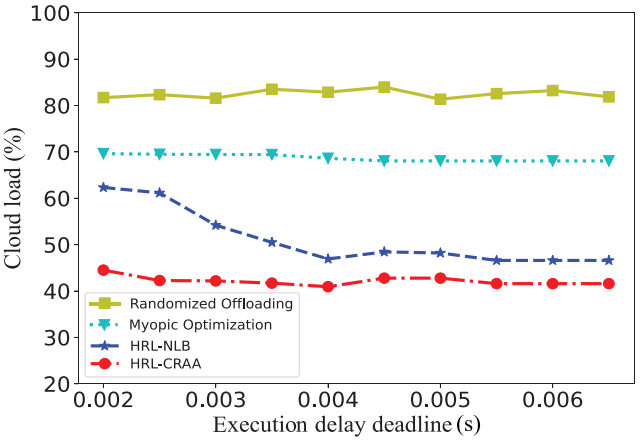

Fig. 9. Cloud load ratio for different execution delay deadlines with different schemes.

We next examine the resource utilization of the proposed cooperative MEC system and present the results from the first to the 15th time slot in Fig. 6. We present the quantity of workload execution of local MEC servers (HRL-CRAA-local) and remote cloud (HRL-CRAA-cloud). Specifically, Fig. 6 shows the ratio of the amount of workloads executed at the MEC servers to the total amount of workloads at the top of each bar. In each time slot, the local MEC servers serve most of the CS workloads, which verify that our proposed HRL-CRAA can achieve efficient utilization of computational resources.

We also present the impact of computational capability F_n of MEC servers on the workloads of cloud execution under four different approaches. In Fig. 7, F_n for each MEC server is increased from 800 to 3000 (in CPU cycles). Note that as F_n is increased from 800 to 2000, the ratio of the amount of cloud execution to the total amount of workloads gradually decreases, except for the randomized offloading approach. When F_n is between 2000 and 3000, the maximum utilization of system computation resources is reached, as the curves of HRL-CRAA and myopic optimization tend to be stabilized. When F_n is equal to 3000, because the MEC servers have enough computational capabilities to process the arrived workloads locally, the curve of HRL-NLB becomes close to that of HRL-CRAA. Especially, compared with the other baseline approaches, our proposed HRL-CRAA scheme achieves a smaller cloud load ratio under the same computation capability, which means that HRL-CRAA executes more workloads at the MEC servers and achieves efficient utilization of computational resources.

The impact of the computational capability on the average system cost is presented in Fig. 8, which shows that as the increase of computation capability F_n of MEC servers, the average system cost will gradually decrease. Combined with the cloud computing ratio results in Fig. 7, compared with myopic optimization, HRL-CRAA achieves a smaller cloud load ratio but a larger averaged system cost. In addition, when F_n is beyond 2000, HRL-CRAA and HRL-NLB have the same average system cost. Thus, when the MEC servers have sufficient computational capabilities to execute the arrived workloads, load balancing among MEC servers will not be very helpful. In other words, our proposed cooperative MEC system and the HRL-CRAA scheme are suitable for MEC systems with limited computation resources and under heavy CS workloads.

Finally, we investigate the impact of the execution delay deadline $D_k^{\max}$ of CS on the workloads of cloud execution and the averaged system cost in Figs. 9 and 10, respectively.

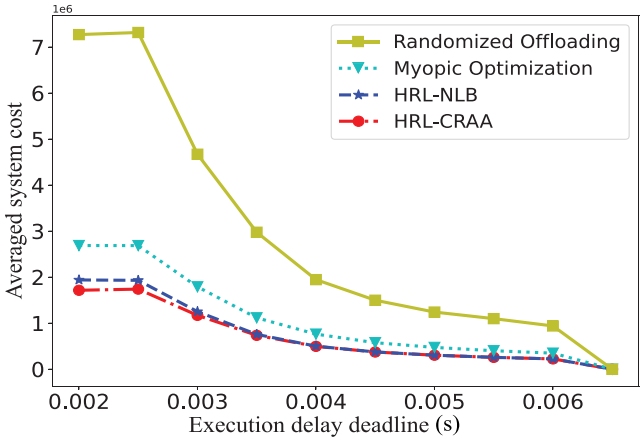

Fig. 10. Averaged system cost for different execution delay deadlines with different schemes.

We increase $D_k^{\max}$ from 2×10^{-3} s to 6.5×10^{-3} s in the experiments. In Fig. 9, when $D_k^{\max}$ is increased from 2×10^{-3} s to 4×10^{-3} s, the cloud load ratios gradually decrease for HRL-CRAA, HRL-NLB, and the myopic optimization scheme. Beyond $D_k^{\max} = 4 \times 10^{-3}$ s, the curves approach their respective stable values. Intuitively, when a larger execution delay deadline, the MEC servers with the same computational capability can serve more workloads until most or all the current arrived workloads are executed. In Fig. 10, we can see that increasing the execution delay deadline $D_k^{\max}$ can effectively reduce the average system costs for all four schemes. Specifically, beyond $D_{\max}^k = 4 \times 10^{-3}$, HRL-NLB and HRL-CRAA have the same average system cost, which means the computational capability of each MEC server is sufficient for executing all the requested CS workloads.

VIII. Conclusion

In this article, we investigated a hierarchical caching and resource allocation problem in a cooperative MEC system with multiple types of CSs. We formulated the problem as a long-term time horizon cost minimization MDP, with a reward function that maximizes the utilization of communication and computation resources while avoiding invalid transmission of workloads. We transformed the problem into two subproblems: i.e., the lower layer and upper layer optimization, and then designed a two-tier DQN to solve these two subproblems. Our simulation results validated that our proposed policy can achieve a superior performance compared to the baseline approaches, as well as converge and scale linearly with the network size, making it suitable for MEC systems with multiple servers and users.

References

[1] Y. Mao, J. Zhang, S. H. Song, and K. B. Letaief, "Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 16, no. 9, pp. 5994–6009, Sep. 2017.

[2] H. Guo, J. Liu, and J. Zhang, "Computation offloading for multi-access mobile edge computing in ultra-dense networks," *IEEE Commun.*, vol. 56, no. 8, pp. 14–19, Aug. 2018.

[3] S. Yu, R. Langar, X. Fu, L. Wang, and Z. Han, "Computation offloading with data caching enhancement for mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11098–11112, Nov. 2018.

[4] B. Yang et al., "Edge intelligence for autonomous driving in 6G wireless system: Design challenges and solutions," *IEEE Wireless Commun.*, vol. 28, no. 2, pp. 40–47, Apr. 2021.

[5] G. Jia, G. Han, J. Du, and S. Chan, "A maximum cache value policy in hybrid memory-based edge computing for mobile devices," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4401–4410, Jun. 2019.

[6] M. Yan, W. Li, C. A. Chan, S. Bian, I. Chih-Lin, and A. F. Gygax, "PECS: Towards personalized edge caching for future service-centric networks," *China Commun.*, vol. 16, no. 8, pp. 93–106, Aug. 2019.

[7] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 12, pp. 3590–3605, Dec. 2016.

[8] B. Yang, X. Cao, X. Li, C. Yuen, and L. Qian, "Lessons learned from accident of autonomous vehicle testing: An edge learning-aided offloading framework," *IEEE Wireless Commun. Lett.*, vol. 9, no. 8, pp. 1182–1186, Aug. 2020.

[9] L. Wang, H. Wu, Y. Ding, W. Chen, and H. V. Poor, "Hypergraph-based wireless distributed storage optimization for cellular D2D underlays," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 10, pp. 2650–2666, Oct. 2016.

[10] Y. Qian, R. Wang, J. Wu, B. Tan, and H. Ren, "Reinforcement learning-based optimal computing and caching in mobile edge network," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 10, pp. 2343–2355, Oct. 2020.

[11] I. Chih-Lin, S. Han, Z. Xu, S. Wang, Q. Sun, and Y. Chen, "New paradigm of 5G wireless Internet," *IEEE J. Sel. Areas Commun.*, vol. 34, no. 3, pp. 474–482, Mar. 2016.

[12] C. Wang, C. Liang, F. R. Yu, Q. Chen, and L. Tang, "Joint computation offloading, resource allocation and content caching in cellular networks with mobile edge computing," in *Proc. IEEE ICC*, Paris, France, May 2017, pp. 1–6.

[13] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, vol. 9, no. 3, pp. 1529–1541, Jul.–Sep. 2021.

[14] X. Sun and N. Ansari, "EdgeIoT: Mobile edge computing for the Internet of Things," *IEEE Commun.*, vol. 54, no. 12, pp. 22–29, Dec. 2016.

[15] X. Cao et al., "Edge-assisted multi-layer offloading optimization of LEO satellite-terrestrial integrated networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 381–398, Feb. 2023.

[16] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online geographical load balancing for energy-harvesting mobile edge computing," in *Proc. IEEE ICC*, Kansas City, MO, USA, May 2018, pp. 1–6.

[17] H. Liao et al., "Cloud-edge-device collaborative reliable and communication-efficient digital twin for low-carbon electrical equipment management," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 1715–1724, Feb. 2023.

[18] Y. Sun, Z. Chen, M. Tao, and H. Liu, "Bandwidth gain from mobile edge computing and caching in wireless multicast systems," *IEEE Trans. Wireless Commun.*, vol. 19, no. 6, pp. 3992–4007, Jun. 2020.

[19] C. Park and J. Lee, "Mobile edge computing-enabled heterogeneous networks," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1038–1051, Feb. 2021.

[20] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[21] N. Abbas, Y. Zhang, A. Taherkordi, and T. Skeie, "Mobile edge computing: A survey," *IEEE Internet Things J.*, vol. 5, no. 1, pp. 450–465, Feb. 2018.

[22] Y. Sun, S. Zhou, and Z. Niu, "Distributed task replication for vehicular edge computing: Performance analysis and learning-based algorithm," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1138–1151, Feb. 2021.

[23] C. You, K. Huang, H. Chae, and B.-H. Kim, "Energy-efficient resource allocation for mobile-edge computation offloading," *IEEE Trans. Wireless Commun.*, vol. 16, no. 3, pp. 1397–1411, Mar. 2017.

[24] S. Yu, X. Chen, Z. Zhou, X. Gong, and D. Wu, "When deep reinforcement learning meets federated learning: Intelligent multitimescale resource management for multiaccess edge computing in 5G ultradense network," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2238–2251, Feb. 2021.

[25] S. Xia, Z. Yao, Y. Li, and S. Mao, "Online distributed offloading and computing resource management with energy harvesting for heterogeneous MEC-enabled IoT," *IEEE Trans. Wireless Commun.*, vol. 20, no. 10, pp. 6743–6757, Oct. 2021.

[26] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.
[27] W. Shi, J. Cao, Q. Zhang, Y. Li, and L. Xu, "Edge computing: Vision and challenges," *IEEE Internet Things J.*, vol. 3, no. 5, pp. 637–646, Oct. 2016.
[28] Z. Zhang and M. Tao, "Deep learning for wireless coded caching with unknown and time-variant content popularity," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1152–1163, Feb. 2021.
[29] L. Chen, J. Xu, and S. Zhou, "Computation peer offloading in mobile edge computing with energy budgets," in *Proc. IEEE GLOBECOM*, Singapore, Dec. 2017, pp. 1–6.
[30] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Trans. Netw.*, vol. 26, no. 4, pp. 1619–1632, Aug. 2018.
[31] T. Zhang, K. Zhu, and J. Wang, "Energy-efficient mode selection and resource allocation for D2D-enabled heterogeneous networks: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 1175–1187, Feb. 2021.
[32] S. Deng, H. Zhao, W. Fang, J. Yin, S. Dustdar, and A. Y. Zomaya, "Edge intelligence: The confluence of edge computing and artificial intelligence," *IEEE Internet Things J.*, vol. 7, no. 8, pp. 7457–7469, Aug. 2020.
[33] S. Shen, T. Zhang, S. Mao, and G.-K. Chang, "DRL-based channel and latency aware radio resource allocation for 5G service-oriented RoF-mmWave RAN," *J. Lightw. Technol.*, vol. 39, no. 18, pp. 5706–5714, Sep. 2021.
[34] X. Shen et al., "AI-assisted network-slicing based next-generation wireless networks," *IEEE Open J. Veh. Technol.*, vol. 1, pp. 45–66, 2020.
[35] C. Zhou et al., "Deep reinforcement learning for delay-oriented IoT task scheduling in SAGIN," *IEEE Trans. Wireless Commun.*, vol. 20, no. 2, pp. 911–925, Feb. 2021.
[36] X. Wang, C. Wang, X. Li, V. C. M. Leung, and T. Taleb, "Federated deep reinforcement learning for Internet of Things with decentralized cooperative edge caching," *IEEE Internet Things J.*, vol. 7, no. 10, pp. 9441–9455, Oct. 2020.
[37] L. Tan, Z. Kuang, L. Zhao, and A. Liu, "Energy-efficient joint task offloading and resource allocation in OFDMA-based collaborative edge computing," *IEEE Trans. Wireless Commun.*, vol. 21, no. 3, pp. 1960–1972, Mar. 2022.
[38] K. Psounis, A. Zhu, B. Prabhakar, and R. Motwani, "Modeling correlations in Web traces and implications for designing replacement policies," *Comput. Netw.*, vol. 45, no. 4, pp. 379–398, Jul. 2004.
[39] K. Poularakis, J. Llorca, A. M. Tulino, I. Taylor, and L. Tassiulas, "Joint service placement and request routing in multi-cell mobile edge computing networks," in *Proc. IEEE INFOCOM*, Paris, France, Apr./May 2019, pp. 10–18.
[40] R. B. Cooper, *Introduction to Queueing Theory*. Amsterdam, The Netherlands: North Holland, 1981.

Wenqian Zhang received the Ph.D. degree in information and communication intelligent system from Donghua University, Shanghai, China, in 2022.

From 2019 to 2020, she was a Visiting Scholar with the Department of Electrical and Computer Engineering, Auburn University, Auburn, AL, USA. She is currently an Assistant Professor with the Department of Communication Engineering, Shanghai Maritime University, Shanghai. Her research interests include edge intelligence, mobile-edge computing, and blockchain.

Guanglin Zhang (Member, IEEE) received the Ph.D. degree in information and communication engineering from Shanghai Jiao Tong University, Shanghai, China, in 2012.

From 2013 to 2014, he was a Postdoctoral Research Associate with the Institute of Network Coding, Chinese University of Hong Kong, Hong Kong. He is currently a Professor and the Department Chair with the Department of Communication Engineering, and he is the Vice Dean of the College of Information Science and Technology, Donghua University, Shanghai. His research interests include capacity scaling of wireless networks, vehicular networks, smart microgrid, and mobile-edge computing.

Prof. Zhang serves as a Technical Program Committee Member for IEEE Globecom 2016–2017, IEEE ICC 2014, 2015, and 2017, IEEE VTC2017-Fall, IEEE/CIC ICCC 2014, WCSP 2014, APCC 2013, and WASA 2012. He serves as the Local Arrangement Chair for ACM TURC 2017 and the Vice TPC Co-Chair for ACM TURC 2018. He serves as an Editor for *China Communications* and *Journal of Communications and Information Networks*.

Shiwen Mao (Fellow, IEEE) received the Ph.D. degree in electrical engineering from Polytechnic University, Brooklyn, NY, USA, in 2004.

After joining Auburn University, Auburn, AL, USA, in 2006, he held the McWane Endowed Professorship from 2012 to 2015 and the Samuel Ginn Endowed Professorship from 2015 to 2020 with the Department of Electrical and Computer Engineering. He is currently a Professor and the Earle C. Williams Eminent Scholar, and the Director of the Wireless Engineering Research and Education Center, Auburn University. His research interests include wireless networks, multimedia communications, and smart grid.

Prof. Mao received the Southeastern Conference 2023 Faculty Achievement Award for Auburn, the IEEE ComSoc MMTC Outstanding Researcher Award in 2023, the IEEE ComSoc TC-CSR Distinguished Technical Achievement Award in 2019, the Auburn University Creative Research and Scholarship Award in 2018, the NSF CAREER Award in 2010, and several service awards from IEEE ComSoc. He is a co-recipient of the 2022 Best Journal Paper Award of IEEE ComSoc eHealth Technical Committee (TC), the 2021 Best Paper Award of Elsevier/KeAi *Digital Communications and Networks* Journal, the 2021 IEEE INTERNET OF THINGS JOURNAL Best Paper Award, the 2021 IEEE Communications Society Outstanding Paper Award, the IEEE Vehicular Technology Society 2020 Jack Neubauer Memorial Award, the 2018 Best Journal Paper Award and the 2017 Best Conference Paper Award from IEEE ComSoc Multimedia TC, and the 2004 IEEE Communications Society Leonard G. Abraham Prize in the Field of Communications Systems. He is a co-recipient of the Best Paper Awards from IEEE ICC 2022 and 2013, IEEE GLOBECOM 2023, 2019, 2016, and 2015, and IEEE WCNC 2015, and the Best Demo Awards from IEEE INFOCOM 2022 and IEEE SECON 2017. He is the Editor-in-Chief of IEEE TRANSACTIONS ON COGNITIVE COMMUNICATIONS AND NETWORKING and an Area Editor of ACM GetMobile. He is the General Chair of IEEE INFOCOM 2022, a TPC Chair of IEEE INFOCOM 2018, the TPC Vice-Chair of IEEE INFOCOM 2015, and the TPC Vice Chair of IEEE GLOBECOM 2022. He is a Distinguished Lecturer of the IEEE Communications Society and the IEEE Council of RFID.