Deep Reinforcement Learning for Energy-Efficient Computation Offloading in Mobile-Edge Computing

Huan Zhou, *Member, IEEE*, Kai Jiang, Xuxun Liu, *Member, IEEE*, Xiuhua Li, *Member, IEEE*, and Victor C. M. Leung, *Life Fellow, IEEE*

***Abstract*—Mobile-edge computing (MEC) has emerged as a promising computing paradigm in the 5G architecture, which can empower user equipments (UEs) with computation and energy resources offered by migrating workloads from UEs to the nearby MEC servers. Although the issues of computation offloading and resource allocation in MEC have been studied with different optimization objectives, they mainly focus on facilitating the performance in the quasistatic system, and seldomly consider time-varying system conditions in the time domain. In this article, we investigate the joint optimization of computation offloading and resource allocation in a dynamic multiuser MEC system. Our objective is to minimize the energy consumption of the entire MEC system, by considering the delay constraint as well as the uncertain resource requirements of heterogeneous computation tasks. We formulate the problem as a mixed-integer nonlinear programming (MINLP) problem, and propose a value iteration-based reinforcement learning (RL) method, named *Q*-Learning, to determine the joint policy of computation offloading and resource allocation. To avoid the curse of dimensionality, we further propose a double deep *Q* network (DDQN)-based method, which can efficiently approximate the value function of *Q*-learning. The simulation results demonstrate that the proposed methods significantly outperform other baseline methods in different scenarios, except the exhaustion method. Especially, the proposed DDQN-based method achieves very close performance with the exhaustion method, and can significantly reduce the average of 20%, 35%, and 53% energy consumption compared with offloading decision, local first method, and offloading first method, respectively, when the number of UEs is 5.**

***Index Terms*—Computation offloading, energy consumptions, mobile-edge computing (MEC), reinforcement learning (RL), resource allocation.**

Manuscript received September 8, 2020; revised February 21, 2021 and April 8, 2021; accepted June 7, 2021. Date of publication June 22, 2021; date of current version January 7, 2022. This work was supported in part by the National Natural Science Foundation of China under Grant 61872221. *(Corresponding author: Xuxun Liu.)*

Huan Zhou and Kai Jiang are with the College of Computer and Information Technology, China Three Gorges University, Yichang 443002, China (e-mail: zhouhuan117@gmail.com; jiangkai0112@gmail.com).

Xuxun Liu is with the Research Centre of Multimedia Information Security Detection and Intelligent Processing, South China University of Technology, Guangzhou 510641, China (e-mail: liuxuxun@scut.edu.cn).

Xiuhua Li is with the School of Big Data and Software Engineering, Chongqing University, and also with the Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Ministry of Education, Chongqing 401331, China (e-mail: lixiuhua1988@gmail.com).

Victor C. M. Leung is with the College of Computer Science and Software Engineering, Shenzhen University, Shenzhen 518060, China, and also with the Department of Electrical and Computer Engineering, The University of British Columbia, Vancouver, BC V6T 1Z4, Canada (e-mail: vleung@ieee.org).

Digital Object Identifier 10.1109/JIOT.2021.3091142

I. Introduction

WITH the progress of technology and the popularity of mobile devices in the emerging fifth-generation (5G) networks, mobile applications, especially increasing computation-intensive applications, such as online interactive games, face recognition, and augmented/virtual reality, have led to an explosive growth of data traffic [1]–[3]. In general, these flourishing applications always have intense requirements on Quality of Service (QoS), which brings higher energy consumption than traditional applications. However, considering the physical size and economic constraints, current user equipments (UEs) have suffered from the limitation of computation resources and energy power, which may become an inevitable bottleneck to address the accompanying challenges in processing extensive applications or providing persistent energy [4], [5].

Mobile cloud computing (MCC) can enable convenient access to a shared resource pool in the centralized cloud. It was once considered as a promising solution to relieve the increasingly apparent conflict between application requests and resource-constrained UEs, considering that the cloud server is deployed with higher computation and storage capacity than UEs [6]. But, indeed, the cloud servers are spatially far from UEs, which make MCC becoming unfeasible for latency-sensitive applications as the long transmission distance in cloud-based processing may contribute to extra cost and delay [7], [8]. Besides, despite the continuous efforts that have been spent on enhancing the channel bandwidth in MCC, the spectrum resource's utilization efficiency is notably reaching its theoretical boundary [9], [10]. Thus, such efforts will not be sufficient, and there often lacks of a reasonable resource assignment scheme to deal with the tremendous service traffic in 5G networks.

All these challenges have expedited the development of mobile-edge computing (MEC). Specifically, MEC is emerged as a crucial component to cope with the computation-intensive tasks in the 5G architecture [11], [12], of which it can move computation, caching, and network functions toward the network edges. Compared with MCC, it extends cloud computing services from the centralized cloud to the edges of networks, and enables UEs to offload workloads to nearby MEC servers directly by leveraging base stations or access points (APs). Such a pattern not only accommodates the expansion requirements of the computation capabilities of UEs but also improves the QoS of mobile applications with considerably reduced latency and energy consumption [13]–[15].

Meanwhile, with the revival of artificial intelligence, deep reinforcement learning (DRL), which combines reinforcement learning (RL) and deep learning, has been regarded as a suitable method to search the asymptotically optimal solutions in time-varying edge environments [16]. Without any prior knowledge, DRL can well capture environments' hidden dynamics by enhancing the intelligence of edge networks, thus learning the strategies to realize the best long-term goal through the repeated interaction within a specific context [17], [18]. This feature makes DRL exhibit its particular potentials when designing computation offloading and resource allocation schemes in a dynamic system.

In this article, we focus on jointly optimizing the offloading decision and resource allocation coordinately for task execution in MEC. Our objective is to minimize the energy consumptions of the entire MEC system. We formulate the corresponding problem as a mixed-integer nonlinear programming (MINLP) problem and propose a value iteration-based RL method, named Q-learning, to determine the joint policy of computation offloading and resource allocation. However, the state and action space may increase exponentially with the increase of UEs' number in Q-learning, leading to maintain the Q-table impossibly because of the curse of dimensionality. To make up this disadvantage, we further propose a double deep Q network (DDQN)-based method, which can efficiently approximate the value function of Q-learning by exploiting deep neural networks (DNNs). The key contributions can be summarized as follows.

1) We explore a dynamic time-varying system, in which the uncertain resource requirements and the delay constraint of heterogeneous computation tasks are both considered.
2) We jointly optimize the computation offloading and resource allocation coordinately in MEC, and formulate the corresponding problem as an MINLP problem to minimize the long-term energy consumption of the entire MEC system.
3) To address the formulated energy consumption minimization problem, we define a Markov decision process (MDP) for the optimization process in the system. Then, we propose a value iteration-based RL method, named Q-learning, and a DDQN-based method, respectively.
4) Simulation results demonstrate that our proposed Q-learning and DDQN-based methods significantly outperform other baselines in different scenarios, except the exhaustion method. Furthermore, the performance of the proposed DDQN-based method is very close to that of the Exhaustion method.

The remainder of this article is organized as follows. First, Section II reviews the related work. Then, we describe the system model in Section III, including the network architecture, communication model, computation model, and energy consumption. In Section IV, we formulate the problem as an MINLP problem with the objective to minimize the energy consumption of the entire MEC system. In Section V, we propose the strategies of computation offloading and resource allocation, i.e., a Q-learning-based method and a DDQN-based method. Furthermore, we give and analyze the simulation results in Section VI. Finally, Section VII concludes this article.

II. Related Work

The research of computation offloading in MEC has attracted widespread concerns from academia and industry. Table I gives a brief comparison of some existing works. For instance, Deng *et al.* [10] made an exhaustive review on computation offloading and discussed its future potentials. Mao *et al.* [19] investigated the joint optimization of offloading strategy and multidimensional resource in a NOMA-assisted MEC system, intending to minimize the total energy consumption. Zhang *et al.* [20] exploited the energy-latency tradeoff in single and multicell MEC network scenarios, and proposed an iterative search algorithm combining the interior penalty function with D.C. programming to find the optimal solution. Furthermore, a dynamic computation offloading and resource allocation method was proposed based on Lyapunov optimization in [21]. For conserving energy and reducing the delay for all users, a separable semidefinite relaxation scheme was adopted in [22] to make similar joint optimization. Chen and Hao [23] leveraged the idea of the software-defined network and investigated the task offloading problem in the ultradense network, where resource competition among UEs is also taken into account. Samanta and Chang [24] designed an adaptive service offloading scheme to improve the resource utilization efficiency in edge networks. Ning *et al.* [25] considered the partial computation offloading, and presented an iterative heuristic resource allocation algorithm, which adaptively adjusts cooperative offloading decisions in the system. In practice, however, the above studies are not suitable for the dynamic system, since most of them only focus on facilitating the quasistatic system's performance, and seldomly consider time-varying system conditions in the time domain. Besides, the uncertain resource requirements, the limited resource capacity, as well as the computation tasks' sensitive latency property in the MEC system are all nontrivial issues, which can affect the performance of computation offloading dramatically.

To cope with the challenges mentioned above, some studies have proposed some DRL-based methods in MEC [26]–[37]. Specifically, Dai *et al.* [26] incorporated action refinement in DRL, and designed a new algorithm to optimize computation offloading and resources allocation concurrently. Zhan *et al.* [27] formulated the offloading problem as a partially observable MDP, and applied the game theory to the designed policy gradient DRL-based method. Huang *et al.* [28] investigated a wireless-powered MEC network and proposed a DRL-based online method, which can achieve near-optimal decisions by considering only a small subset of candidate actions in each iteration. Feng *et al.* [29] developed a cooperative computation offloading and resource allocation framework for blockchain-enabled MEC systems, and proposed an asynchronous advantage actor-critic-based method to achieve the optimal tradeoff between the performance of blockchain

TABLE I
COMPARISON OF EXISTING WORKS FOCUSING ON COMPUTATION OFFLOADING

Task Model	**Resource Optimization**		**Optimization Objective**	**Additional Constraint**	**Reference**	**Proposed Solution**
	Comp.[1]	Comm.[1]				
Binary Offloading	✗	✗	Economical overhead	Task execution deadline	[8]	Greedy, dynamic programming based method
	✗	✗			[24]	An adaptive algorithm using the Lagrange multiplier
	✓	✓			[34]	A DQN based method
	✓	✗			[36]	A DDQN based distributed method
	✓	✓		Network stability	[13]	Lyapunov optimization theory
	✓	✓		Battery capacities, task execution deadline	[39]	Lyapunov optimization theory
	✗	✗			[2]	A game theoretic approach
	✓	✗	The tradeoffs between execution latency and energy consumption	Task execution deadline	[3]	A game theoretic approach
	✓	✓			[6]	A location-based method in the cooperative scenario
	✓	✗			[35]	A DRL framework based on the actor-critic structure, and a low-complexity algorithm for the critic network
	✓	✓		Battery capacities	[20]	An iterative search algorithm combining interior penalty function with D.C. programming
	✓	✗			[30]	State Action Reward State Action (SARSA)
	✓	✓			[22]	An efficient three-step algorithm comprising of semidefinite relaxation, alternating optimization, and sequential tuning
	✓	✗			[32]	Q-learning based and DRL based methods
	✓	✓	Energy consumption	Task execution deadline	[38]	A reformulation linearization-technique-based Branch-and-Bound method, and a Gini coefficient-based greedy heuristic
	✓	✓			[40]	Energy-efficient MEC cloud offloading and radio resource allocation schemes
	✓	✗			[26]	Deep Deterministic Policy Gradient (DDPG)
	✓	✓		Power capacity of the UE, task execution deadline	[31]	An online algorithm based on DRL
	✓	✓		Task execution deadline	Our paper	Q-learning based and DDQN based methods
	✓	✗			[7]	Sequential convex programming algorithm, Knapsack method
	✓	✗	Execution latency	Battery capacities	[23]	An efficient software defined task offloading scheme
	✓	✗			[33]	A model-free Q-learning based method
	✗	✓	System capacity and spectrum efficiency	The QoS demands of UEs	[18]	A resource allocation with apolicy gradient method
	✗	✓	The weighted sum of the computation rate	Energy constraint of UEs	[28]	A DRL-based online offloading framework
	✓	✗		Task execution deadline	[29]	An asynchronous advantage actor–critic-based algorithm
	✓	✓	The long-term utility of the network		[37]	Q-learning based method and DRL method
Partial Offloading	✓	✗	The tradeoffs between execution latency and energy consumption		[16]	A tide ebb algorithm
	✓	✓		The system total cost	[27]	A decentralized approach based on DRL with policy gradient and differential neural computer
	✓	✓	Energy consumption	Task execution deadline	[19]	Block coordinate descent method, and a resource allocation algorithm
	✓	✗	Execution latency	Interference among multiple UEs	[25]	An iterative heuristic method

and MEC. Alfakih *et al.* [30] proposed an RL-based state–action–reward–state–action algorithm to resolve the resource management problem in the edge server. In [31], the joint optimization of the multitask computation offloading, NOMA transmission, and computation-resource allocation was performed for the time-varying channel realizations. Similarly, Li *et al.* [32] achieved a deep Q network (DQN)-based adaptive framework to tackle the computation offloading and resource allocation in MEC without any prior knowledge. The framework was also used by Pan *et al.* [33] to study the problem of dependency-aware task offloading decisions, which aims to optimize the delay of applications. In [34], an online DRL-based resource allocation method was exploited, considering UEs' mobility in a diverse multitimescale framework. Furthermore, Yan *et al.* [35] proposed a low-complexity critic network to jointly determine the offloading and resource allocation decisions under time-varying wireless fading channels and stochastic edge computing capability. Tang and Wong [36] incorporated the long short-term memory and dueling DQN to improve the estimation of the long-term cost in the computation offloading algorithm. Liu *et al.* [37] considered the scenario that moving vehicles can provide computation services for UEs in a vehicle edge computing network, and proposed a DRL method to seek optimal computation offloading and resource allocation policies.

Inspired by existing studies, this article considers the tasks' delay constraint, and minimizes energy consumption of the entire MEC system. Meanwhile, except for making the offloading decision of each UE, allocating resources on demand is also an essential factor for achieving the benefit of computation offloading in MEC. We jointly consider the optimization of the offloading decision, spectrum resource allocation, and computation resource allocation in this article. As shown in Table I,[1] although some studies have considered the optimization objective of energy consumption, few studies have considered

[1]Comp. means the computation resource and Comm. means the communication resource.

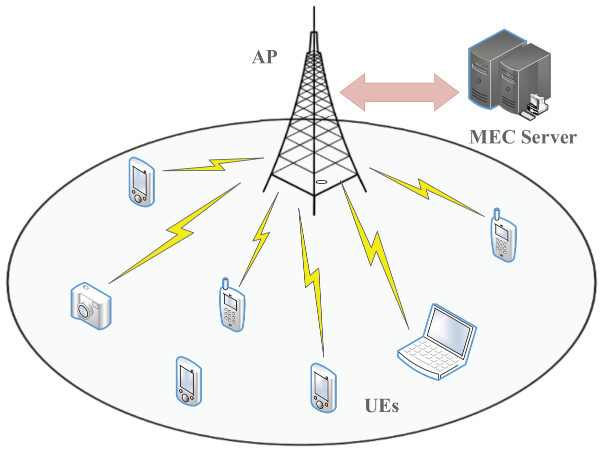


Fig. 1. Multiuser mobile edge network model.

these three aspects jointly in a dynamic multiuser MEC system to the best of our knowledge. Furthermore, we develop a Q-learning and a DDQN-based method, respectively, to address the NP-hard optimization problem in this article, where the DDQN-based method can reduce the observed overestimations compared with the widely adopted DRL-based methods.

III. System Model

In this section, the system model of the multiuser MEC network is presented. We first introduce the network architecture adopted in this article, and then give the detail of the analytical model, including the communication model, computation model, and energy consumption. The illustration of main notations is summarized in Table II.

A. Network Architecture

We premeditate a single-cell scenario with an AP and n UEs, as shown in Fig. 1. The set of UEs is denoted by $\mathcal{N} = \{1, 2, \ldots, n\}$. In order to provide computation services for UEs, an MEC server is deployed at the AP, and UEs can determine whether their computation tasks need to be offloaded to the MEC server via the wireless link. Furthermore, we assume that the system operates in fixed length of time slots $t \in \{0, 1, 2, \ldots, \tau\}$, where τ denotes the finite time horizon. In each time slot, each UE will generate only one computation-intensive task. The computation task is conceived as atomic and cannot be divided into partitions, which means that the computation task may need to be offloaded to the MEC server or executed locally. When multiple tasks need to be offloaded concurrently, the MEC server manager should allocate the spectrum and computation resource in the system, based on the time-varying system conditions, heterogeneity of tasks, and energy overhead of the entire system.

Without loss of generality, we adopt a widely used model to describe the arrived computation task on a certain UE i ($i \in \mathcal{N}$) in time slot t, which can be denoted by three terms as $\Lambda_i^t \triangleq \{s_i^t, c_i^t, d_{i,\max}^t\}$. Here, s_i^t stands for the data size of the computation task and c_i^t reflects the amount of computation resource required to accomplish the task. For example, c_i^t can be quantified as the total number of CPU cycles required to process the task Λ_i^t. $d_{i,\max}^t$ denotes the maximum tolerant delay of the task Λ_i^t, which means that the task execution time should not exceed $d_{i,\max}^t$, whether the task is executed either locally or by computation offloading.

Moreover, we assume that UEs are randomly scattered in the coverage scope of the AP and request edge services during a relatively long period. Notably, we only concentrate on executing locally or offloading the task to the MEC server in this article, without further offloading to the remote cloud or other base stations. The decision of any UE i for the arrived task Λ_i^t can be represented by the integer variable $x_i^t \in \{0, 1\}$ in time slot t, where $x_i^t = 0$ means the task is processed by UE i's CPU locally, and $x_i^t = 1$ indicates the task is offloaded to the MEC server. Consequently, we can define the offloading decision vector of the MEC system in time slot t as $\boldsymbol{\mathcal{X}}(\boldsymbol{t}) = \{x_1^t, x_2^t, \ldots, x_n^t\}$.

B. Communication Model

Within the coverage of AP, each UE can offload the computation task to the deployed MEC server when the task is hard to be executed locally under restrained conditions. Here, we ignore the communication overhead between the MEC server and AP. Since there is only one AP in a cell, the overlapping coverage between neighboring cells and the interference among UEs is also not considered. Now, we assume if multiple UEs decide to offload their tasks to the AP simultaneously, the bandwidth is assigned to each UE based on its demands by using the technique of DSA. We denote the spectrum resource allocation vector of all UEs by $\boldsymbol{\mathcal{B}}(\boldsymbol{t}) = \{\theta_1^t, \theta_2^t, \ldots, \theta_n^t\}$, where $\theta_i^t \in [0, 1]$ is the percentage of the spectrum allocated to UE i in time slot t. Hence, the achievable uplink transmission rate of UE i is calculated as [32] and [38]

$$R_i^t = \theta_i^t W \log_2\left(1 + \frac{p_i g_i}{\theta_i^t W N_0}\right) \tag{1}$$

where W denotes the channel bandwidth of the available spectrum. p_i is the transmission power of UE i when uploading its task to the AP, N_0 represents the noise power spectral density, and g_i is the channel gain of the wireless propagation channel between UE i and the AP.[2]

C. Computation Model

The computation task Λ_i^t can be executed either locally on UE i's own computation resources or on the MEC server via computation offloading. Next, we introduce these two computation models in detail.

1) Local Execution Model: For $x_i^t = 0$, the task Λ_i^t will be processed locally. In reality, different UEs may have distinct computation capabilities (in CPU cycles per second) and energy consumptions per CPU cycle. We use f_i^{loc} and δ_i^{loc} to denote these two metrics of UE i, respectively. Thus, the

[2]To simplify the communication model, we assume that the UEs' mobility is not fast during the task offloading, so channel gains can be considered a constant for UEs, while they may be different for different UEs.

execution time of task Λ_i^t in such case can be given as

$$D_{i,\mathrm{loc}}^t = \frac{c_i^t}{f_i^{\mathrm{loc}}} \tag{2}$$

and the corresponding energy consumption of UE i is calculated as

$$E_{i,\mathrm{loc}}^t = c_i^t \delta_i^{\mathrm{loc}} \tag{3}$$

where $\delta_i^{\mathrm{loc}} = 10^{-27}(f_i^{\mathrm{loc}})^2$, hinging on the practical chip architecture, which is referred from measurement in [39].

2) Mobile Edge Execution Model: For $x_i^t = 1$, UE i decides to offload its computation task Λ_i^t to the MEC server in time slot t. After handling the computation task, the MEC server returns the computation results back to UE i. Note that we neglect the transmission delay and energy consumption for sending back the computation results from the MEC server to UEs, due to the tiny data size and the higher downlink rate in most instances. Therefore, the total processing time of the task Λ_i^t mainly contains two parts: the first part is the time for transmitting the task Λ_i^t from UE i to the MEC server, and the second part is the computation execution time on the MEC server.

The time for uploading the task Λ_i^t is associated with the input data size and the uplink transmission rate of UE i directly, so we have

$$d_{i,\mathrm{trans}}^t = \frac{s_i^t}{R_i^t}. \tag{4}$$

Then, the corresponding energy consumption for transmitting the task Λ_i^t to the MEC server can be calculated as

$$E_{i,\mathrm{trans}}^t = p_i d_{i,\mathrm{trans}}^t = \frac{p_i s_i^t}{R_i^t}. \tag{5}$$

On the other hand, we denote the computation resource allocation vector in time slot t by $\mathcal{C}(t) = \{\beta_1^t, \beta_2^t, \ldots, \beta_n^t\}$, where $\beta_i^t \in [0, 1]$ is the percentage of computation resources assigned to complete the task Λ_i^t by the MEC server. As the entire computation resource of the MEC server is f_{MEC}; hence, $\beta_i^t f_{\mathrm{MEC}}$ represents the computation resource allocated to UE i in time slot t. When a higher proportion of computation resources is allocated to a certain UE, the task execution time becomes smaller, but the energy consumption may increase. Meanwhile, the variable β_i^t must satisfy the constraint of $\sum_{i=1}^{\mathcal{N}} x_i^t \beta_i^t \leq 1$. Therefore, the execution time of the task Λ_i^t on the MEC server is given by

$$d_{i,\mathrm{exe}}^t = \frac{c_i^t}{\beta_i^t f_{\mathrm{MEC}}}. \tag{6}$$

When the MEC server processes the offloaded computation task Λ_i^t in time slot t, the corresponding energy consumption of the MEC server can be expressed as [40]

$$E_{i,\mathrm{MEC}}^t = P_{\mathrm{MEC}} \beta_i^t d_{i,\mathrm{exe}}^t \tag{7}$$

where P_{MEC} is the power of the MEC server when it operates at full capacity.

In addition, when the MEC server executes the offloaded computation tasks, UEs are supposed to wait for the return outcomes. We assume that UEs operate in the standby mode during this period, and the power consumption of UE i in the standby state is p_i^w. Thus, the corresponding energy consumption of UE i is calculated as

$$E_{i,\mathrm{idle}}^t = p_i^w d_{i,\mathrm{exe}}^t = \frac{p_i^w c_i^t}{\beta_i^t f_{\mathrm{MEC}}}. \tag{8}$$

Accordingly, combining the above calculating process, the overall delay and energy consumption for UE i to offload the task Λ_i^t are easily formulated as

$$D_{i,\mathrm{offl}}^t = d_{i,\mathrm{trans}}^t + d_{i,\mathrm{exe}}^t = \frac{s_i^t}{R_i^t} + \frac{c_i^t}{\beta_i^t f_{\mathrm{MEC}}} \tag{9}$$

and

$$\begin{aligned} E_{i,\mathrm{offl}}^t &= E_{i,\mathrm{MEC}}^t + E_{i,\mathrm{trans}}^t + E_{i,idle}^t \\ &= P_{\mathrm{MEC}} \beta_i^t d_{i,\mathrm{exe}}^t + \frac{p_i s_i^t}{R_i^t} + \frac{p_i^w c_i^t}{\beta_i^t f_{\mathrm{MEC}}}. \end{aligned} \tag{10}$$

D. Energy Consumption

In each time slot t, UE i is supposed to make one decision to execute the task Λ_i^t, so the execution time of the task can be expressed as

$$\begin{aligned} D_i^t &= \left(1 - x_i^t\right) D_{i,\mathrm{loc}}^t + x_i^t D_{i,\mathrm{offl}}^t \\ &= \left(1 - x_i^t\right) \frac{c_i^t}{f_i^{\mathrm{loc}}} + x_i^t \left(\frac{s_i^t}{R_i^t} + \frac{c_i^t}{\beta_i^t f_{\mathrm{MEC}}}\right). \end{aligned} \tag{11}$$

TABLE II
NOTATIONS AND SYMBOLS

Notation	Explanation
$\mathcal{N}$	The set of all UEs
Λ_i^t	The arrived computation task on UE i in time slot t
s_i^t	The size of the task Λ_i^t
c_i^t	The number of CPU cycles required to process the task Λ_i^t
$d_{i,max}^t$	The maximum tolerant delay of the task Λ_i^t
x_i^t	The offloading decision of UE i in time slot t
p_i	The transmission power of UE i
θ_i^t	The percentage of the spectrum allocated to UE i in time slot t
R_i^t	The uplink channel transmission rate of UE i in time slot t
f_i^{loc}	The computation capability of UE i
δ_i^{loc}	The energy consumption per CPU cycle of UE i
$D_{i,loc}^t$	The execution time of the task Λ_i^t by local execution model
$E_{i,loc}^t$	The energy consumption of the task Λ_i^t by local execution model
$d_{i,trans}^t$	The time for transmitting Λ_i^t from UE i to the MEC server
$E_{i,trans}^t$	The energy consumption for uploading the task Λ_i^t
β_i^t	The percentage of the computation resources allocated to complete the task Λ_i^t by the MEC server
$d_{i,exe}^t$	The time for processing the task Λ_i^t by the MEC server
p_i^w	UE i's power in the standby state
$E_{i,idle}^t$	The energy consumption of UE i during waitting the return outcome
$D_{i,offl}^t$	The delay of the task Λ_i^t by computation offloading
$E_{i,offl}^t$	The energy consumption of the task Λ_i^t by computation offloading
D_i^t	The delay of the task Λ_i^t
E_i^t	The energy consumption of the task Λ_i^t
E_{total}^t	The energy consumption of the entire system in time slot t

Similarly, in order to accomplish the task Λ_i^t in time slot t, the energy consumption of the task is

$$\begin{aligned}E_i^t &= \left(1-x_i^t\right)E_{i,\mathrm{loc}}^t + x_i^t E_{i,\mathrm{offl}}^t \\ &= \left(1-x_i^t\right)c_i^t\delta_i^{\mathrm{loc}} + x_i^t\left(P_{\mathrm{MEC}}\beta_i^t d_{i,\mathrm{exe}}^t + \frac{p_i s_i^t}{R_i^t} + \frac{p_i^w c_i^t}{\beta_i^t f_{\mathrm{MEC}}}\right).\end{aligned} \tag{12}$$

Considering the computation tasks of all UEs in time slot t, we can formulate the total energy consumption of the entire MEC system in time slot t as

$$\begin{aligned}E_{\mathrm{total}}^t &= \sum_{i=1}^{\mathcal{N}} E_i^t \\ &= \sum_{i=1}^{\mathcal{N}}\left[\left(1-x_i^t\right)E_{i,\mathrm{loc}}^t + x_i^t E_{i,\mathrm{offl}}^t\right].\end{aligned} \tag{13}$$

IV. Problem Formulation

In this article, we investigate the joint optimization problem of computation offloading and resource allocation in the proposed MEC system. Our objective is to minimize the long-term energy consumption of the entire MEC system under specified delay constraints of the tasks. To this end, the corresponding optimization problem can be formulated as

$$\begin{aligned}\min_{\boldsymbol{\mathcal{X}}(t),\boldsymbol{\mathcal{B}}(t),\boldsymbol{\mathcal{C}}(t)} \quad & \lim_{\tau\to\infty}\frac{1}{\tau}\sum_{t=0}^{\tau-1}E_{\mathrm{total}}^t \\ s.\ t. \quad & C1\colon\ x_i^t \in \{0,1\} \quad \forall i\in\mathcal{N} \\ & C2\colon\ D_i^t \le d_{i,\max}^t \quad \forall i\in\mathcal{N} \\ & C3\colon\ 0 \le \sum_{i=1}^{\mathcal{N}} x_i^t\beta_i^t \le 1 \quad \forall i\in\mathcal{N} \\ & C4\colon\ 0 \le \beta_i^t \le 1 \quad \forall i\in\mathcal{N} \\ & C5\colon\ 0 \le \sum_{i=1}^{\mathcal{N}} x_i^t\theta_i^t \le 1 \quad \forall i\in\mathcal{N} \\ & C6\colon\ 0 \le \theta_i^t \le 1 \quad \forall i\in\mathcal{N}.\end{aligned} \tag{14}$$

Here, $\lim_{\tau\to\infty} 1/\tau \sum_{t=0}^{\tau-1} E_{\mathrm{total}}^t$ is the time averaged energy consumption of the entire MEC system. The meaning of the above constraints is as follows.

1) C1 denotes that each UE only chooses local or edge execution model to process the computation task.
2) C2 guarantees that the execution time of the task should not exceed the maximum tolerant delay.
3) C3 and C6 are the constraints of the total available computation and spectrum resources, i.e., the resources allocated to all offloaded UEs cannot exceed the total amount of resources provided by the MEC server.
4) C4 ensures that the computation resources allocated to UE i cannot exceed the total amount of computation resources provided by the MEC server.
5) C7 ensures that the spectrum resources allocated to UE i cannot exceed the total available spectrum resources of the AP.

To address the above problem in (14), it is necessary to find the optimal results of offloading decision vector $\boldsymbol{\mathcal{X}}(t) = \{x_i^t | i \in \mathcal{N}\}$, computation resource allocation vector $\boldsymbol{\mathcal{C}}(t) = \{\beta_i^t | i \in \mathcal{N}\}$, and spectrum resource allocation vector $\boldsymbol{\mathcal{B}}(t) = \{\theta_i^t | i \in \mathcal{N}\}$ in each time slot, which can be coordinated to minimize the overall computation energy consumption in the system with the given delay constraints. Notably, the offloading decision variables x_i^t are binary variables, while the resource allocation variables β_i^t and θ_i^t are dynamically changing. The system needs to collect a large amount of network state information and make the global decision on computation offloading and resource allocation based on the network's current state. Moreover, we devote to a more practical case in which the prior information of task request pattern within a period is unknown in advance. Therefore, the objective function is an MINLP problem and undoubtedly NP-hard. The feasible set of the problem is not convex, and the complexity always increases exponentially with the increase of the number of UEs. Since conventional model-based methods may not be able to adapt to the dynamic property and make intelligent decisions, we propose RL-based methods to address the considered problem in this article.

V. Computation Offloading and Resource Allocation Optimization Problem Using Reinforcement Learning

In this section, we approximate the optimization problem as an MDP to minimize the energy consumption of the entire system. First, we define the state, action, and reward function in MDP. Second, we introduce the widely used Q-learning-based method, and then propose a DDQN-based method to avoid the curse of dimensionality, which can estimate the value function of Q-learning by utilizing DNNs.

A. State, Action, and Reward Definition

Three critical elements in MDP are identified as follows.

1) *State Space:* The state in MDP is a space to reflect the network environment. Accordingly, in our proposed system, the state of the available resources at time slot t is determined by the realization of the states $1 - \sum_i^{\mathcal{N}} x_i^t\theta_i^t$, $1-\sum_{i=1}^{\mathcal{N}} x_i^t\beta_i^t$, where the former is the percentage of the available computation resources of the MEC server, and the latter is the percentage of the available spectrum resources at present. The purpose of observing them is to keep the constraint of computation capacity and communication channel capacity. Furthermore, we also need to observe the energy consumption E_{total}^t in each time slot to compare whether the optimal state is reached. Hence, the state vector at time slot t can be described as $z_t = (E_{\mathrm{total}}^t, 1 - \sum_i^{\mathcal{N}} x_i^t\theta_i^t, 1 - \sum_{i=1}^{\mathcal{N}} x_i^t\beta_i^t)$.
2) *Action Space:* The objective of the agent is to map the space of states to the space of actions. In our proposed MEC system, the agent is responsible to decide the offloading strategies of various UEs in each time slot. Besides, it also need to determine the corresponding percentage of the resources allocated to UEs in each time slot t. Therefore, the action vector consists of

three parts: a) the offloading decision vector $\boldsymbol{\mathcal{X}(t)} = \{x_1^t, x_2^t, \ldots, x_n^t\}$; b) the computation resource allocation vector $\boldsymbol{\mathcal{C}(t)} = \{\beta_1^t, \beta_2^t, \ldots, \beta_n^t\}$; and c) the spectrum resource allocation vector $\boldsymbol{\mathcal{B}(t)} = \{\theta_1^t, \theta_2^t, \ldots, \theta_n^t\}$ of all UEs, respectively. So we can present the current action vector as $\boldsymbol{d_t} = \{\boldsymbol{\mathcal{X}(t)}, \boldsymbol{\mathcal{C}(t)}, \boldsymbol{\mathcal{B}(t)}\}$, with the combination of some possible values of these three parts.

3) *Reward Function:* In general, the immediate network reward function is related to the objective function. In the considered optimization problem, we aim to minimize the energy consumption of the entire MEC system, which is the inverse goal of RL that tries to achieve the maximum cumulative discounted reward. Thus, the value of reward needs to be negatively correlated to the value of the objective function. The agent obtains $r(\boldsymbol{z_t}, \boldsymbol{d_t})$ in state $\boldsymbol{z_t}$ when action $\boldsymbol{d_t}$ is performed at time slot t. To minimize the long-term energy consumption of the entire MEC system, we define the immediate reward as normalized $r(\boldsymbol{z_t}, \boldsymbol{d_t}) = -E(\boldsymbol{z_t}, \boldsymbol{d_t})$, where $E(\boldsymbol{z_t}, \boldsymbol{d_t}) = E_{\text{total}}^t$ is the actual total energy consumption of the current state.

The agent's reward value should satisfy the constraint conditions to ensure the validity of the calculation results. If constraints in (14) are violated, we set the reward as a punitive negative directly. Besides, to reduce the action space as much as possible, we introduce a preclassification step before the learning process. For UE i, if the execution time under local processing exceeds the preset threshold $d_{i,\max}^t$, UE i will be scheduled to offload its task to the MEC server, and the corresponding decision x_i^t will be fixed to 1 in the time slot t, as well as the allocated computation resources should satisfy $\beta_i^t f_{\text{mec}} \geq c_i/(d_{i,\max}^t - s_i^t/R_i^t)$. In such a way, the possible values of offloading decision and resource allocation can be reduced to limit the agent's action space.

B. Markov Decision Process

Almost all programming problems that are controlled by decision making can be described in terms of MDP. Generally, MDP can be solved by linear programming or dynamic programming methods. However, when the transition probability and the immediate reward are time varying and unknown in discrete time steps, the RL-based methods are more suitable to deal with these problems.

In this article, we approximate the optimization problem as an MDP, where the agent adaptively learns the optimal behavior through the repeated interaction within a unknown environment. Specifically, the agent observes current state of the environment as $z_t \in \mathcal{Z}$, and then selects and executes an admissible action $d_t \in \mathcal{D}$ according to a policy π in each time step, where $\mathcal{Z}, \mathcal{D}$ are the finite state and action space, respectively. The policy $\pi : \mathcal{Z} \to p(\mathcal{D} = d|\mathcal{Z})$ is defined as a mapping from the current state to the corresponding action, for which a specific policy $\pi(z_t, d_t)$ can guide the decision action d_t under different states z_t. After that, the agent will transfer into the next new state z_{t+1} with the transition probability $p(z_{t+1}|z_t, d_t)$, and obtains an immediate reward $r_t = r(z_t, d_t)$.

For the long-term consideration, a state value function $V^\pi(z_t)$ is defined by the cumulative discounted reward value $R_t = \sum_{t=0}^\infty \varphi^t r_t$ for agent in the state z_t, which can be used to indicate the long-term effect by making a policy π in the current state (i.e., measuring the quality of a certain action or an available state–action pair). Thus, under the policy $\pi(z_t)$, the recursive $V^\pi(z_t)$ based on any initial state z_0 can be expressed as follows:

$$V^\pi(z_t) = \mathbb{E}_\pi[R_t|z_0 = z_t] = \mathbb{E}_\pi\left[\left(\sum_{t=0}^{\infty} \varphi^t r_t\right) \mid z_0 = z_t\right] \tag{15}$$

where $\mathbb{E}$ denotes the expectation, and $\varphi \in (0, 1)$ is the discounting factor indicating the importance of the predicted future rewards.

We now use $z_{t+1} \in \mathcal{Z}$ to represent the next state after executing an action d_t at the current state z_t, and the transition probability from z_t to z_{t+1} is given as $p(z_{t+1}|z_t, d_t)$. As we have formulated the system environment as an MDP, the state value function $V^\pi(z_t)$ can be transformed to the temporal difference form according to the *Bellman equation*

$$\begin{aligned} V^\pi(z_t) &= \mathbb{E}_\pi\left[\left(\sum_{t=0}^{\infty} \varphi^t r_t\right)|z_0 = z_t\right] \\ &= \mathbb{E}_\pi\left[\left(r(z_t, d_t) + \sum_{t=1}^{\infty} \varphi^t r_t\right)|z_0 = z_{t+1}\right] \\ &= r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1}|z_t, d_t) V^\pi(z_{t+1}). \end{aligned} \tag{16}$$

According to the process above, the goal of the agent is to make an optimal control policy $\pi^*(z_t) \to d_t^* \in \mathcal{D}$, so that it can obtain the exepcted maximum cumulative discounted reward under the current state z_t. Therefore, the optimization problem in this article can be converted to an optimal state value function $V^{\pi^*}(z_t)$, which is expressed as

$$\begin{aligned} V^{\pi^*}(z_t) &= \max_\pi \left[r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1}|z_t, d_t) V^{\pi^*}(z_{t+1}) \right] \\ \text{s.t.} \quad & \text{constraints in (C1)} - \text{(C6)}. \end{aligned} \tag{17}$$

Thus, the solution $\pi^*(z_t)$, which satisfies (17), is guaranteed to become the optimal policy, and the optimal action d_t^* for state z_t is easily obtained as

$$d_t^* = \underset{d_t}{\operatorname{argmax}}\ V^\pi(z_t, d_t). \tag{18}$$

C. Q-Learning-Based Solution

Q-learning is an effective model-free RL algorithm based on value iteration, in which both environment and the state transition probability are not explicit. Q-learning enables the agent to adaptively learn the optimal behavior separately within a specific context in each time step. It continually approximates the optimal Q value instead of modeling the dynamic knowledge of MDP. Specifically, the agent of Q-learning needs to compute the result of Q function[3] of each state–action pair,

[3]Note that we do not distinguish the concept of "state–action function" and "Q-function" in the rest of this article.

and updates the Q value in a maintained dimension Q-table after each interaction. Hereafter, the corresponding policy is derived by selecting the action with the highest Q value under each state.

We adopt the Q-learning-based method to address the proposed Markov decision problem. Particularly, the Q-learning-based method can estimate the optimal action values $Q(z, d)$ of the state and admissible action pairs for each time step, which are stored or updated in a Q-table. Here, we apply the state–action function $Q(z_t, d_t)$ to estimate the optimal state value function $V^{\pi^*}(z_t)$ under the state z_t, and the relationship between them is

$$V^{\pi^*}(z_t) = \max_{d_t} Q^{\pi}(z_t, d_t). \tag{19}$$

Therefore, the expected cumulative return after conducting an action d_t can be expressed as

$$Q^{\pi}(z_t, d_t) = r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1}|z_t, d_t) V^{\pi^*}(z_{t+1}). \tag{20}$$

Then, we can rewrite (20) with the following form by incorporating: (19) with (20)

$$Q^{\pi}(z_t, d_t) = r(z_t, d_t) + \varphi \sum_{z_{t+1}} p(z_{t+1}|z_t, d_t) \max_{d_{t+1}} Q^{\pi}(z_{t+1}, d_{t+1}). \tag{21}$$

Hereafter, we try to iteratively update the state–action function by using the recursive method at each time step. The relevant problem is then reduced to find the optimal policy π^* by merely identifying the action that maximizes $Q^{\pi}(z_t, d_t)$ in each time slot. Theăcoreăof Q-learningăis the process of value iteration, and the iterative formula of Q value can be obtained as follows:

$$Q(z_t, d_t)^{\text{new}} = Q(z_t, d_t) + \varepsilon\left[r(z_t, d_t) + \varphi \max_{d_{t+1}} Q(z_{t+1}, d_{t+1}) - Q(z_t, d_t)\right] \tag{22}$$

where $\varepsilon \in (0, 1)$ is the learning rate parameter. The Q value can definitely converge to the optimal value $Q^{\pi^*}(z_t, d_t)$ when an appropriate ε is designated.

More details of the proposed Q-learning-based method are summarized in Algorithm 1. In the multiuser MEC scenario, each UE has no way of knowing the information about others except leveraging repeating observation of the system, and the system gradually learns to update actions with the corresponding functions to optimize the configuration of various offloading decisions and dynamic resource allocation. Our objective is to minimize the energy consumption of the entire MEC system by allocating optimal resource allocation variables under various offloading decisions. The agent in Q-learning learns and makes decisions separately through the repeated interaction, considering the Q value as a long-term cumulative reward. Initially, the Q value for each state and action pair is set randomly as stated by line 1 in Algorithm 1. It is worth noticing that the optimal policy π^* from the Q-table sometimes has the disadvantage of being vulnerable to the limited search area, which heavily depends on the quality and quantity of the training data. In order to provide the tradeoff between exploitation and exploration in RL, we can choose the action d_t with the ϵ-greedy strategy, where ϵ is a diminishing value to provide exploring. Specifically, the UE chooses the action that maximizes the Q value with the probability $1 - \epsilon$ (exploiting), and the other action with the tiny probability ϵ (exploring), which can be expressed as follows:

$$d_t = \begin{cases} d, \epsilon \\ \text{argmax}_d Q(z_t, d), 1 - \epsilon. \end{cases} \tag{23}$$

After conducting an action d_t, the environment grants the agent a reward $r(z_t, d_t)$, and the current state z_t will transfer to the next state z_{t+1}. With this information, the corresponding Q value for a state–action pair then can be updated using (22).

Algorithm 1 Q-Learning-Based Algorithm for Computation Offloading and Resource Allocation

```
Input: state space $\mathcal{Z}$, action space $\mathcal{D}$, learning rate $\varepsilon$, discount
    factor $\varphi$
Output: the $Q$ values for every state-action pair
 1: Initialize $Q(z, d)$ arbitrarily for $\forall\ z \in \mathcal{Z},\ d \in \mathcal{D}$
 2: for each episode do
 3:     Initialize $z_t$
 4:     for each steps of episode do
 5:         Choose an action with a random probability $\Phi$ in the
            current $z_t$ as
 6:         if $\Phi \le \epsilon$ then
 7:             randomly select an action $d_t$
 8:         else
 9:             select $d_t = \text{argmax}_d Q(z_t, d)$
10:         end if
11:         Execute action $d_t$, observe the reward $r_t$ and the next
            state $z_{t+1}$
12:         Update new $Q(z_t, d_t)$: $= Q(z_t, d_t) + \varepsilon\Big[r(z_t, d_t) +$
            $\varphi\max_{d_{t+1}} Q(z_{t+1}, d_{t+1}) - Q(z_t, d_t)\Big]$
13:         Let $z_t \leftarrow z_{t+1}$;
14:         Terminate When the expected state $z_{terminal}$ reaches
15:     end for
16: end for
```

D. DDQN-Based Solution

Although the best policy can be obtained by recording and updating the Q value in the Q table constantly, it is easy to be trapped in the curse of dimensionality as the possible action–state space may be enormous in practical time-varying scenarios. Besides, if we record the corresponding Q value with this form, it will take a long time to search the corresponding value in such a big table, and the memory may not be enough to maintain the table.

To avoid this bottleneck in the Q-learning-based method, we further propose an advanced DDQN-based method, in which the DNNs are adopted to estimate the action-value function of Q-learning. DDQN is an improvement of the traditional DQN algorithm. Normally, DQN uses the same Q network to evaluate the actions and approximate the action-value function.

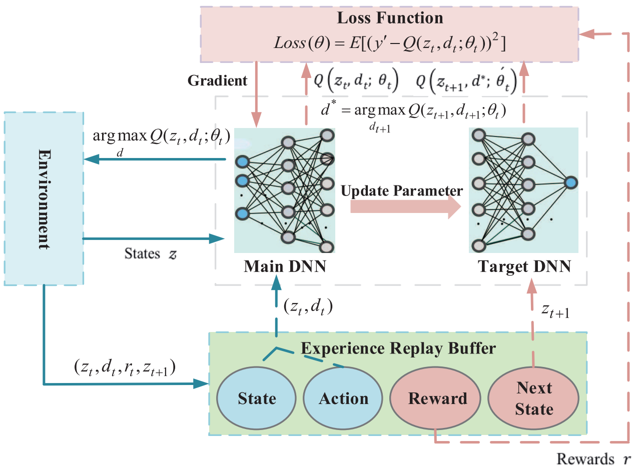


Fig. 2. Training process of DDQN.

However, this training process is easy to perform overestimation of the action value function, which may have adverse impacts on the real value. Therefore, the basic idea of DDQN is to separate the selection and evaluation of actions by constructing different action functions.

As mentioned above, we can calculate the value function for arbitrary state–action pair by using (22) and create a lookup table to represent and update it in the Q-learning-based method. In contrast, the DDQN-based method can approximately obtain the optimal Q value by using the updated DNN parameter θ. The Q value in DDQN can be expressed as follows:

$$Q(z,d)\approx Q(z,d;\theta) \tag{24}$$

where θ is the weight of the main neural network, and there exists another target neural network, which will be described later.

Compared to the run-of-mill Q-learning-based method, an experience replay buffer is utilized in the DDQN-based method, for which a DDQN agent stores its transition memory tuple $(z_t, d_t,\ r_t,\ z_{t+1})$ into the replay buffer at each time step t. Meanwhile, the arrived samples can be used to train the neural network's parameters later, and the agent will randomly select a minibatch sample from the experience replay buffer to train the DNN's parameters. That is to say, we can randomly select some previous experiences for learning in each update. Some studies indicate that the experience replay can improve sample efficiency and break the correlation among data training, while accelerating the convergence rate of DDQN [41], [42]. On the other hand, there exists a fixed Q-target mechanism in DDQN, where two neural networks with the same structure but different parameters are maintained to disrupt the pertinence. The target neural network aims to acquire the temporal difference target Q value while the main neural network can evaluate the Q function. Note that the weight parameters $\acute{\theta}_j$ of target DNN are updated periodically by the counterpart θ_j of the main DNN by $\acute{\theta}=\zeta\theta+(1-\zeta)\acute{\theta}$ with $\zeta\ll 1$. Then, the

Algorithm 2 DDQN-Based Algorithm for Computation Offloading and Resource Allocation

```
1:  Initialize the experience replay buffer
2:  Initialize the main deep neural network with random weight $\theta$
3:  Initialize the target deep neural network with $\acute{\theta}=\theta$
4:  for each episode do
5:     Obtain the initial observation $s_1$, and pre-process $s_1$ as the beginning state $z_1$
6:     for each steps of episode do
7:        Choose $d_t$ with a random probability $\Phi$ as
8:        if $\Phi\le\epsilon$ then
9:           randomly select an action $d_t$
10:       else
11:          select $d_t=\text{argmax}_d(z_t,d_t;\theta)$
12:       end if
13:       Execute action $d_t$ in the MEC system, receive the reward $r_t$ and the next observation $s_{t+1}$
14:       Pre-process $s_{t+1}$ to be the next state $z_{t+1}$
15:       Store the experience $(z_t,d_t,\ r_t,\ z_{t+1})$ into the experience replay buffer
16:       Randomly select a minibatch of $R$ sample $(z_j,d_j,\ r_j,\ z_{j+1})$ from the experience replay buffer
17:       Calculate the target $Q$ value as
          $\acute{y}_j=r_j+\varepsilon Q\Big(z_{j+1},\text{argmax}_{d_{j+1}}Q(z_{j+1},d_{j+1};\theta_j);\acute{\theta}_j\Big)$
18:       Update the weight of the main deep neural network by minimizing the Loss($\theta$) as
          $\text{Loss}(\theta)=\frac{1}{R}\sum_j\big(\acute{y}_j-Q(z_j,d_j;\theta_j)\big)^2$
19:       Perform a gradient descent step on Loss($\theta$) with respect to the parameter $\theta$ by $\frac{\partial\text{Loss}(\theta)}{\partial\theta}$
20:       Update the target deep neural network parameter $\acute{\theta}_j$ by using weight $\theta_j$
21:    end for
22: end for
```

fixed Q-target is used to generate the target Q value $\acute{y}$ as

$$\acute{y}_j=r_j+\varepsilon Q\Big(z_{j+1},\text{argmax}_{d_{j+1}}Q(z_{j+1},d_{j+1};\theta_j);\acute{\theta}_j\Big). \tag{25}$$

Moreover, the target Q network updates the weight after some training steps, instead of updating it at each time step. By doing so, the learning process becomes more stable.

Throughout the training process, the DDQN agent randomly selects a minibatch of R sample $(z_j,d_j,\ r_j,\ z_{j+1})$ from the experience replay buffer to eliminate the coupling of time-series training data. At each iteration, we train the deep Q-function toward the target value by minimizing the loss function Loss(θ), which can be written as

$$\begin{aligned}\text{Loss}(\theta)=&\ \mathbb{E}\bigg[\Big(r_j(z_j,d_j)+\varepsilon Q\Big(z_{j+1},\text{argmax}_{d_{j+1}}Q(z_{j+1},d_{j+1};\theta_j)\\&\qquad\acute{\theta}_j\Big)-Q(z_j,d_j;\theta_j)\Big)^2\bigg]\\=&\ \frac{1}{R}\sum_j\big(\acute{y}_j-Q(z_j,d_j;\theta_j)\big)^2.\end{aligned} \tag{26}$$

Fig. 2 shows the training process of DDQN and more specific detail can be found in Algorithm 2. First, the experience replay buffer and the weights of convolutional neural networks are initialized. Especially, the target DNN is initialized with the same weight to the main DNN, i.e., $\acute{\theta} = \theta$. Then, the DNN is developed to obtain the correlation between each state–action pair (z, d) and its value function $Q(z, d)$. Specifically, we have to preprocess the offloading decision and resource allocation of the MEC system with random policy for a sufficiently long time. Then, we achieve and store the corresponding estimated $Q(z, d)$ with some state transition profiles into an experience replay buffer. Finally, the DNN is pretrained with input state–action pairs (z, d) and outcomes $Q(z, d)$. Henceforth, we can obtain the action selection and Q value. Particularly, in each episode, the DDQN agent first obtains the initial observation of the MEC system and preprocesses the observation as the beginning state z_1. Then, the ϵ-greedy strategy is leveraged again to choose the execution action d_t. With a probability $\Phi \le \epsilon$, an action from the action set $\mathcal{D}$ is randomly chosen. Otherwise, the action is selected with the highest estimated Q value derived from the main neural network with the input of state–action pair (z_t, d_t). Then, we execute the action d_t and obtain the corresponding reward r_t and the next observation z_{t+1} from the MEC system. The transition memory tuple $(z_t, d_t,\ r_t,\ z_{t+1})$ will be stored into the replay buffer to train the DNNs parameters at each time step t. Throughout the training process, a minibatch of R samples is randomly selected by the experience replay mechanism in DDQN. After calculating the target Q value $\acute{y}_j$ in line 17, the DDQN agent updates parameters θ of the main DNN by minimizing $\text{Loss}(\theta)$ and a gradient guiding updates of θ can be calculated by $[(\partial\text{Loss}(\theta))/\partial\theta]$. Hence, stochastic gradient descent is performed until the state–action Q-function converges to the optimal value.

VI. Numerical Results

In this section, extensive simulations are conducted to illustrate the performance of the proposed methods in dynamic MEC scenarios.

A. Simulation Settings

We consider a small cell in radius, where one AP with MEC servers is located at the center. At each time slot, UEs with computation tasks are randomly scattered within the AP's coverage. Here, we consider that different UEs perform distinct computation capabilities, which are uniformly distributed between 0.5 and 1.5 GHz. The MEC system can allocate channel resources based on UEs' demands by using DSA technology. Similar to [23], the channel power gain g_i is modeled as $127 + 30\log(L)$, where L is the distance between UE and MEC. At each time slot, the data size of computation tasks (in Mbit) is uniformly distributed in the range of [16, 80], and the corresponding number of required CPU cycles (in Megacycles) obeys uniform distribution in the range of [1000, 3500]. Here, in the proposed methods, we set the capacity of experience replay buffer $C = 500$, and the capacity of the selected minibatch sample $U = 32$. The learning rate parameter is set as $\varepsilon = 0.1$ and the reward decay as $\varphi = 0.9$. The key evaluation parameters are listed in Table III.

TABLE III
Simulation Parameters

Parameter	Definition	Value
p_i	The transmission power of UE i	23dBm
N_0	The noise power spectral density	-158dBm/Hz
g_i	The channel power gain	$127 + 30log(L)$
W	The transmission bandwidth	10MHz
f_i^{loc}	The computation capability of UE i	[0.5, 1.5]GHz
f_{MEC}	The computation capability of the MEC server	8GHz
P_{MEC}	The full-load power of the MEC server	5W
p_i^w	The standby power of UE i	0.03W
$d_{i,max}^t$	The maximum tolerant delay of the task Λ_i^t	3s

For performance comparison, we introduce the following three benchmark methods in the scenarios of multiple UEs.

1) *Local First:* UEs try to execute their tasks locally as possible under the maximum tolerant delay.
2) *Offloading First:* UEs offload their tasks to the MEC server preferentially, and the whole communication and computation resources of the MEC server are allocated equally to each UE in this method.
3) *Offloading Decision:* The offloading decisions are optimized to minimize the energy consumption of the entire MEC system, without considering the optimization of resource allocation. The resources of the MEC server are allocated equally to each offloaded UE.
4) *Exhaustion:* The exhaustion method is obtained by brute-force search, which is approximately the optimal solution.

It is worth noting that the resource requirements of computation tasks are different and dynamic at each time slot, and a certain UE may be incapable of executing the arrived task locally due to the excessive required computation resources under the limitation of the maximum tolerant delay. Therefore, the key difference between our proposed methods and the baseline methods is that our proposed methods can make offloading decisions and allocate computation resources dynamically.

B. Performance Comparison

In this part, we illustrate the convergence performance of the proposed Q-learning-based method and DDQN-based method, and then we compare the performance of these two proposed methods with the fiducial methods in terms of the energy consumption of the system and the average delay of the task at each time slot.

Fig. 3 shows the convergence performance of the proposed Q-learning-based method and DDQN-based method when the number of UEs in the system is 5. It can be observed that for these two RL-based methods, the system's energy consumption of each episode decreases rapidly with the increase of training episodes, and the efficient computation offloading policies can be successfully learned as the interaction continues. Overall, the DDQN-based method performs better as it converges to a relatively stable value after running about 90 training episodes, which achieves a faster running time than

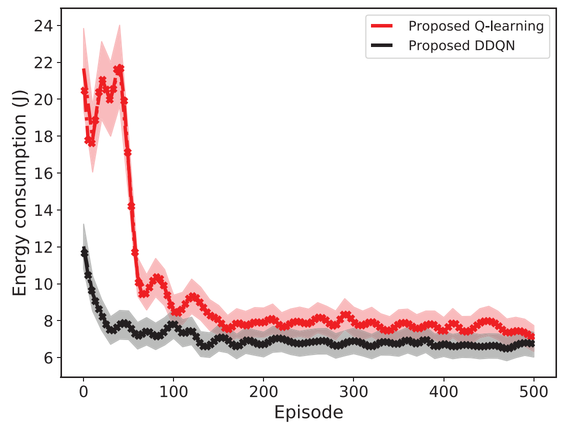


Fig. 3. Convergence of the proposed Q-learning-based method and DDQN-based method.

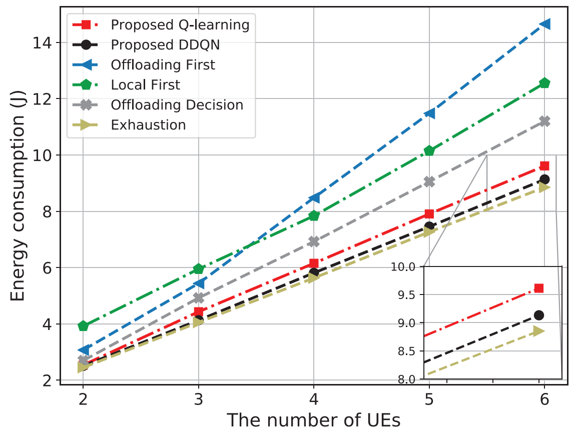


Fig. 4. Energy consumption versus the number of UEs.

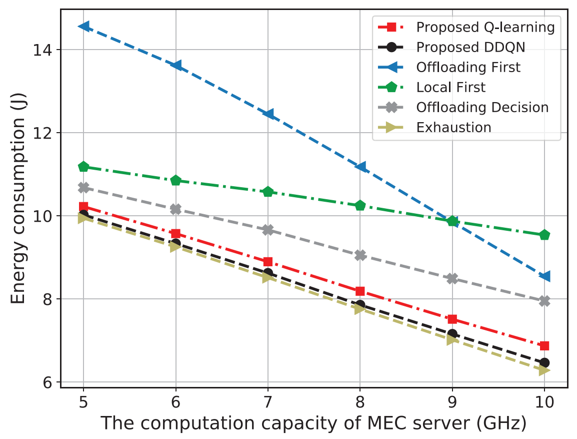


Fig. 5. Energy consumption versus the computation capacity of the MEC server.

the Q-learning-based method. Besides, the total energy consumption of the Q-learning-based method is always slightly higher than that of the DDQN-based method in most episodes, which means that the DDQN-based method is more effective in the offloading process.

Fig. 4 displays the performance comparison of the six methods in terms of the total energy consumption with the increase of the number of UEs, where the computation capability of the MEC server is 8 GHz. The energy consumptions of the six methods all increase accordingly with the increase of the number of UEs. Notably, we can find that the exhaustion method performs as an upper bound to all other methods certainly. It is approximately the optimal solution, which depends on the degree of discretization of resource allocation decisions. However, this method cannot be implemented if the network scale is large. Among the other methods, it can be observed that the proposed DDQN-based method achieves very close performance with the exhaustion method. Furthermore, the proposed DDQN-based method can greatly reduce an average of 20%, 35%, and 53% energy consumption compared with offloading decision, local first method, and offloading first method, respectively, when the number of UEs is 5. Besides, the energy consumption of Q-learning-based method is also smaller than these three fiducial methods, which can be inferred that our proposed methods turn out to be effective under different numbers of UEs. Furthermore, the energy consumption of the offloading first method exceeds the local first method when the number of UEs is 4 and continues to grow substantially with the increase of the number of UEs. This is because the computation and communication resource budget get relatively tight when more tasks need to be executed at each time slot. Once the available resource allocated to a single UE decreases, the delay of transmission and computation for UE will increase significantly, as well as the energy consumption. Compared with the offloading decision method, the proposed Q-learning-based and DDQN-based methods can solve this problem better, because they can dynamically allocate the communication and computation resources to each UE at each time slot.

In the following, we change the MEC server's computation capacity to compare the total energy consumption of different methods. We vary the MEC server's computation capacity from 5 to 10 (in GHz) and the number of UEs in the system is 5. As shown in Fig. 5, the increasing computation capacity of the MEC server has positive impacts on the metric of the energy consumption. As expected, the proposed two methods still perform well under this situation. It can be found that even when the computation capacity of the MEC server in the system is small (5 GHz), the proposed Q-learning-based and DDQN-based method can still reduce 5%, 10%, 43% and 7%, 12%, and 46% energy consumption compared with offloading decision, local first method, and offloading first method, respectively. Furthermore, the offloading first method has a much higher energy consumption than other methods in the initial stage. The difference between the offloading first method and other methods becomes smaller with the increase of the MEC server's computation capacity. The main reason is that as the computation capacity of the MEC server increases, more computation resources can be allocated to UEs; thus, the computation delay will decrease significantly, as well as the energy consumption.

Next, we compare the energy consumption for different amounts of required computation resources by tasks (the same for all tasks in a time slot), where the number of UEs is 5. We change the amount of required computation resources from

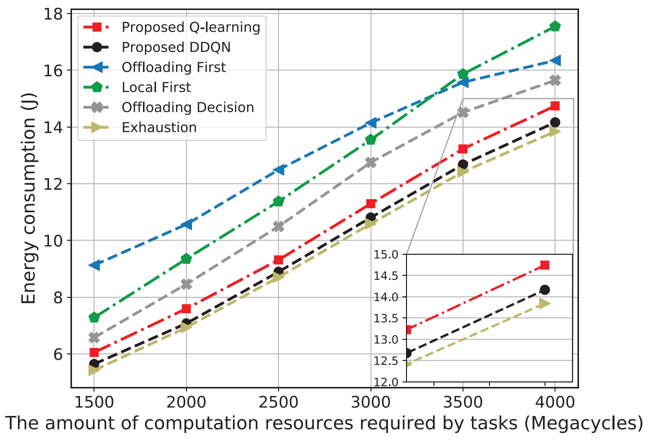


Fig. 6. Energy consumption versus the number of required CPU cycles.

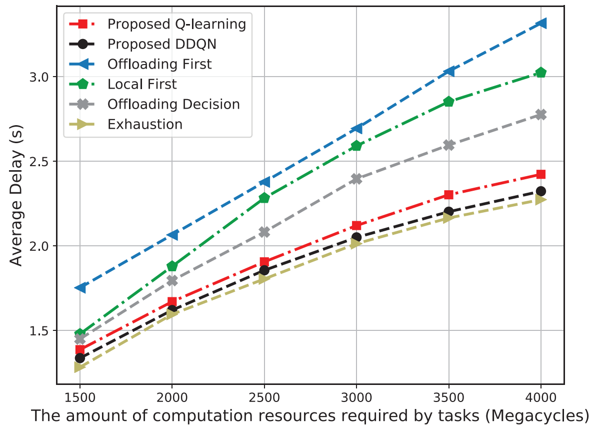


Fig. 7. Average delay versus the number of CPU cycles required by tasks.

1500 to 4000 Megacycles and display the results in Fig. 6. It can be observed that the energy consumption grows with the increase of the computation resource required by tasks for all methods. As expected, the proposed Q-learning-based and DDQN-based methods are always superior to other methods in spite of the exhaustion method. Specifically, they outperform the offloading decision method with the reduction on energy consumption approximately 19% and 26%, respectively, when the amounts of required computation resources are up to 4000 Megacycles, not mentioning the simple rule-based method offloading first and local first. Besides, for the offloading first method, its energy consumption is always much higher than other methods, which means that the resource for offloading five tasks simultaneously in a time slot is very tight under the current system condition. Then, as the amount of required computation resources by tasks increases constantly, it can be found that the energy consumption in the local first exceeds that of the offloading first. This is because the maximum tolerant delay may not be satisfied by computing locally for certain UEs when the required computation resources of task increase. Hereafter, those tasks have to be offloaded to the MEC server under this situation.

Similarly, for the task's average delay, the curves' trends are exactly the same with the results in Fig. 4. As shown in Fig. 7, the average delay of all methods grows with the increase of the amount of required computation resources. The average delay of the offloading first method is always larger than other five methods, due to the relatively limited resource of the MEC server. Furthermore, the proposed DDQN-based method still performs well, as its average delay is smallest except the exhaustion method. Also, the results of the Q-learning-based method are very close to that of the DDQN-based method, which also demonstrates the analysis mentioned above, and that our proposed methods are superior to other fiducial methods.

To summarize, the proposed Q-learning-based and DDQN-based methods can not only reduce the energy consumption effectively but also achieve desirable average delay of all computation tasks under different scenarios. Therefore, we demonstrate that the proposed Q-learning-based and DDQN-based methods are effective under different scenarios.

VII. Conclusion

In this article, we have investigated the joint optimization of computation offloading and resource allocation in a multiuser MEC system, with different resource requirements and time-varying system conditions. Meanwhile, the delay constraint and the limited resource capacity have also been considered. We formulated the optimization problem as a delay constraint energy consumption minimization problem, and used an MDP to describe the relationship between the offloading and resource allocation policies and the system environment. Then, we proposed a Q-learning-based method to obtain the optimal strategy. To avoid the curse of dimensionality caused by the exponential increase of state–action space, we proposed a DDQN-based method. The numerical results demonstrated the effectiveness of the proposed methods under different scenarios. In the future, we plan to exploit a more complex MEC system, in which the offloading tasks can be divided into different partitions.

References

[1] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo, and J. Zhang, "Edge intelligence: Paving the last mile of artificial intelligence with edge computing," *Proc. IEEE*, vol. 107, no. 8, pp. 1738–1762, Aug. 2019.

[2] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[3] K. Zhang, S. Leng, Y. He, S. Maharjan, and Y. Zhang, "Mobile edge computing and networking for green and low-latency Internet of Things," *IEEE Commun. Mag.*, vol. 56, no. 5, pp. 39–45, May 2018.

[4] E. Wang, D. Li, B. Dong, H. Zhou, and M. Zhu, "Flat and hierarchical system deployment for edge computing systems," *Future Gener. Comput. Syst.*, vol. 105, pp. 308–317, Apr. 2020.

[5] G. Premsankar, M. D. Francesco, and T. Taleb, "Edge computing for the Internet of Things: A case study," *IEEE Internet Things J.*, vol. 5, no. 2, pp. 1275–1284, Apr. 2018.

[6] Y. Chao, Y. Liu, X. Chen, and S. L. Xie, "Efficient mobility-aware task offloading for vehicular edge computing networks," *IEEE Access*, vol. 7, pp. 26652–26664, 2019.

[7] B. Liu, C. Liu, and M. Peng, "Resource allocation for energy-efficient MEC in NOMA-enabled massive IoT networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 4, pp. 1015–1027, Apr. 2021, doi: 10.1109/JSAC.2020.3018809.

[8] H. Zhou, X. Chen, S. He, J. Chen, and J. Wu, "DRAIM: A novel delay-constraint and reverse auction-based incentive mechanism for WiFi offloading," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 4, pp. 711–722, Apr. 2020.

[9] H. Li, K. Ota, and M. Dong, "Deep reinforcement scheduling for mobile crowdsensing in fog computing," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–18, 2019.

[10] 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.

[11] 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.

[12] S. Wang, X. Zhang, Y. Zhang, L. Wang, J. Yang, and W. Wang, "A survey on mobile edge networks: Convergence of conputing caching and communications," *IEEE Access*, vol. 5, pp. 6757–6779, 2017.

[13] J. Du, L. Zhao, J. Feng, X. Chu, and F. R. Yu, "Economical revenue maximization in cache enhanced mobile edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Jul. 2018, pp. 1–6.

[14] X. Chen, W. Li, S. Lu, Z. Zhou, and X. Fu, "Efficient resource allocation for on-demand mobile-edge cloud computing," *IEEE Trans. Veh. Technol.*, vol. 67, no. 9, pp. 8769–8780, Sep. 2018.

[15] X. Hou, Y. Li, M. Chen, D. Wu, D. Jin, and S. Chen, "Vehicular fog computing: A viewpoint of vehicles as the infrastructures," *IEEE Trans. Veh. Technol.*, vol. 65, no. 6, pp. 3860–3873, Jun. 2016.

[16] W. Zhang, Z. Zhang, S. Zeadally, H.-C. Chao, and V. C. M. Leung, "MASM: A multiple-algorithm service model for energy-delay optimization in edge artificial intelligence," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4216–4224, Jul. 2019.

[17] Y. Li, X. Wang, X. Gan, H. Jin, L. Fu, and X. Wang, "Learning-aided computation offloading for trusted collaborative mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 12, pp. 2833–2849, Dec. 2020.

[18] D. Wang, H. Qin, B. Song, and X. Du, "Resource allocation in information-centric wireless networking with D2D-enabled MEC: A deep reinforcement learning approach," *IEEE Access*, vol. 7, pp. 114935–114944, 2019.

[19] S. Mao, S. Leng, and Y. Zhang, "Joint communication and computation resource optimization for NOMA-assisted mobile edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, 2019, pp. 1–6.

[20] J. Zhang *et al.*, "Energy-latency tradeoff for energy-aware offloading in mobile edge computing networks," *IEEE Internet Things J.*, vol. 5, no. 4, pp. 2633–2645, Aug. 2018.

[21] Z. Chang, L. Liu, X. Guo, and Q. Sheng, "Dynamic resource allocation and computation offloading for IoT fog computing system," *IEEE Trans. Ind. Informat.*, vol. 17, no. 5, pp. 3348–3357, May 2021.

[22] M.-H. Chen, B. Liang, and M. Dong, "Joint offloading and resource allocation for computation and communication in mobile cloud with computing access point," in *Proc. IEEE Int. Conf. Comput. Commun. (INFOCOM)*, Apr. 2017, pp. 1863–1871.

[23] 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.

[24] A. Samanta and Z. Chang, "Adaptive service offloading for revenue maximization in mobile edge computing with delay-constraint," *IEEE Internet Things J.*, vol. 6, no. 2, pp. 3864–3872, Apr. 2019.

[25] Z. Ning, P. Dong, X. Kong, and F. Xia, "A cooperative partial computation offloading scheme for mobile edge computing enabled Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4804–4814, Jun. 2019.

[26] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Edge intelligence for energy-efficient computation offloading and resource allocation in 5G beyond," *IEEE Trans. Veh. Technol.*, vol. 69, no. 10, pp. 12175–12186, Oct. 2020.

[27] Y. Zhan, S. Guo, P. Li, and J. Zhang, "A deep reinforcement learning based offloading game in edge computing," *IEEE Trans. Mobile Comput.*, vol. 69, no. 6, pp. 883–893, Jun. 2020.

[28] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, vol. 19, no. 11, pp. 2581–2593, Nov. 2020.

[29] J. Feng, F. R. Yu, Q. Pei, X. Chu, J. Du, and L. Zhu, "Cooperative computation offloading and resource allocation for blockchain-enabled mobile-edge computing: A deep reinforcement learning approach," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6214–6228, Jul. 2020.

[30] T. Alfakih, M. M. Hassan, A. Gumaei, C. Savaglio, and G. Fortino, "Task offloading and resource allocation for mobile edge computing by deep reinforcement learning based on SARSA," *IEEE Access*, vol. 8, pp. 54074–54084, 2020.

[31] L. Qian, Y. Wu, F. Jiang, N. Yu, W. Lu, and B. Lin, "NOMA assisted multi-task multi-access mobile edge computing via deep reinforcement learning for Industrial Internet of Things," *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5688–5698, Aug. 2021, doi: 10.1109/TII.2020.3001355.

[32] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.

[33] S. L. Pan, Z. Zhang, Z. Zhang, and D. Zeng, "Dependency-aware computation offloading in mobile edge computing: A reinforcement learning approach," *IEEE Access*, vol. 7, pp. 134742–134753, 2019.

[34] L. T. Tan and R. Q. Hu, "Mobility-aware edge caching and computing in vehicle networks: A deep reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 10190–10203, Nov. 2018.

[35] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Wireless Commun.*, vol. 19, no. 8, pp. 5404–5419, Aug. 2020.

[36] M. Tang and V. W. S. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Trans. Mobile Comput.*, early access, Nov. 10, 2020, doi: 10.1109/TMC.2020.3036871.

[37] Y. Liu, H. M. Yu, S. L. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[38] P. Zhao, H. Tian, C. Qin, and G. Nie, "Energy-saving offloading by jointly allocating radio and computational resources for mobile edge computing," *IEEE Access*, vol. 5, pp. 11255–11268, 2017.

[39] 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.

[40] K. Zhang *et al.*, "Energy-efficient offloading for mobile edge computing in 5G heterogeneous networks," *IEEE Access*, vol. 4, pp. 5896–5907, 2016.

[41] N. C. Luong, D. T. Hoang, S. Gong, D. Niyato, and I. K. Dong, "Applications of deep reinforcement learning in communications and networking: A survey," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3133–3174, 4th Quart., 2019.

[42] T. He, N. Zhao, and H. Yin, "Integrated networking, caching, and computing for connected vehicles: A deep reinforcement learning approach," *IEEE Trans. Veh. Technol.*, vol. 67, no. 1, pp. 44–55, Jan. 2018.

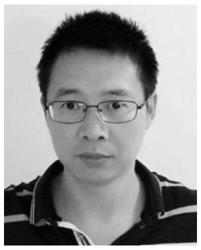
Huan Zhou (Member, IEEE) received the Ph.D. degree from the Department of Control Science and Engineering, Zhejiang University, Hangzhou, China, in 2014.

He was a visiting scholar with the Temple University, Philadelphia, PA, USA, from November 2012 to May 2013, and a CSC supported Postdoctoral Fellow with the University of British Columbia, Vancouver, BC, Canada, from November 2016 to November 2017. He is currently a Full Professor with the College of Computer and Information Technology, China Three Gorges University, Yichang, China. He has published over 50 research papers in some international journals and conferences, including IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, and IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY. His research interests include mobile social networks, VANETs, opportunistic mobile networks, and mobile data offloading.

Prof. Zhou receives the Best Paper Award of I-SPAN in 2014 and I-SPAN in 2018. He is currently serving as an Associate Editor for IEEE ACCESS and *EURASIP Journal on Wireless Communications and Networking*. He was a Lead Guest Editor of *Pervasive and Mobile Computing*, and the Special Session Chair of the Third International Conference on Internet of Vehicles in 2016, and the TPC member of IEEE WCSP'13'14, CCNC'14'15, ICNC'14'15, ANT'15'16, IEEE Globecom'17'18, and ICC'18'19.

Kai Jiang received the B.Sc. degree in measurement and control technology and instrument from Yangtze University, Jingzhou, China, in 2018. He is currently pursuing the M.S. degree in computer technology with China Three Gorges University, Yichang, China.

His research interests include mobile-edge computing and reinforcement learning.



Xuxun Liu (Member, IEEE) received the Ph.D. degree in communication and information systems from Wuhan University, Wuhan, China, in 2007.

He is currently a Professor with the School of Electronic and Information Engineering, and with the Engineering Research Center of Body Data Perception of the Ministry of Education of China, South China University of Technology, Guangzhou, China. He has authored or coauthored over 40 scientific papers in international journals and conference proceedings. His current research interests include wireless sensor networks, wireless body area networks, vehicle *ad hoc* networks, and mobile computing.

Prof. Liu was a recipient of the National Innovation Award of Industry-University Research Collaboration in 2017. He serves as an Associate Editor for the IEEE ACCESS, and as the workshop chair, the publication chair, or the TPC member of a number of conferences.

Xiuhua Li (Member, IEEE) received the B.S. degree from the Honors School, Harbin Institute of Technology, Harbin, China, in 2011, the M.S. degree from the School of Electronics and Information Engineering, Harbin Institute of Technology in 2013, and the Ph.D. degree from the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, BC, Canada, in 2018.

In 2019, he joined Chongqing University, Chongqing, China, through One-Hundred Talents Plan of Chongqing University. He is currently a tenure-track Assistant Professor with the School of Big Data and Software Engineering, Chongqing University, and also the Dean of the Institute of Intelligent Software and Services Computing Associated with the Key Laboratory of Dependable Service Computing in Cyber Physical Society (Chongqing University), Education Ministry, Chongqing. His current research interests are 5G/B5G mobile Internet, mobile-edge computing and caching, big data analytics, and machine learning.

Victor C. M. Leung (Life Fellow, IEEE) received the B.A.Sc. (Hons.) degree in electrical engineering and the Ph.D. degree in electrical engineering from the University of British Columbia (UBC), Vancouver, BC, Canada, in 1977 and 1982, respectively.

He attended graduate school with UBC on a Canadian Natural Sciences and Engineering Research Council Postgraduate Scholarship. From 1981 to 1987, he was a Senior Member of Technical Staff and a Satellite System Specialist with MPR Teltech Ltd., Burnaby, BC, Canada. In 1988, he was a Lecturer with the Department of Electronics, Chinese University of Hong Kong, Hong Kong. He returned to UBC as a Faculty Member in 1989, where he is currently a Professor and a TELUS Mobility Research Chair in Advanced Telecommunications Engineering with the Department of Electrical and Computer Engineering. He has coauthored over 1000 journal/conference papers, 37 book chapters, and co-edited 12 book titles. His research interests are in the broad areas of wireless networks and mobile systems.

Prof. Leung was awarded the APEBC Gold Medal as the head of the graduating class in the Faculty of Applied Science, the IEEE Vancouver Section Centennial Award and 2011 UBC Killam Research Prize, and the 2017 Canadian Award for Telecommunications Research. He is a co-author of the paper that has won the 2017 IEEE ComSoc Fred W. Ellersick Prize. Several of his papers had been selected for best paper awards. He was a Distinguished Lecturer of the IEEE Communications Society. He has served on the editorial boards for the IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, *Wireless Communications Series*, *Series on Green Communications and Networking*, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, IEEE TRANSACTIONS ON COMPUTERS, and *Journal of Communications and Networks*. He is serving on the editorial boards for the IEEE WIRELESS COMMUNICATIONS LETTERS, IEEE TRANSACTIONS ON GREEN COMMUNICATIONS AND NETWORKING, IEEE ACCESS, COMPUTER COMMUNICATIONS, and several other journals. He has guest-edited many journal special issues, and provided leadership to the organizing committees and technical program committees of numerous conferences and workshops. He is a Registered Professional Engineer in the Province of British Columbia, Canada. He is a Fellow of the Royal Society of Canada, the Engineering Institute of Canada, and the Canadian Academy of Engineering.