Computation Offloading for Tasks With Bound Constraints in Multiaccess Edge Computing

Kexin Li, *Student Member, IEEE*, Xingwei Wang, Qiang He, *Associate Member, IEEE*, Qiang Ni, *Senior Member, IEEE*, Mingzhou Yang, and Schahram Dustdar, *Fellow, IEEE*

***Abstract*—Multiaccess edge computing (MEC) provides task offloading services to facilitate the integration of idle resources with the network and bring cloud services closer to the end user. By selecting suitable servers and properly managing resources, task offloading can reduce task completion latency while maintaining the Quality of Service (QoS). Prior research, however, has primarily focused on tasks with strict time constraints, ignoring the possibility that tasks with soft constraints may exceed the bound limits and failing to analyze this complex task constraint issue. Furthermore, considering additional constraint features makes convergent optimization algorithms challenging when dealing with such complex and high-dimensional situations. In this article, we propose a new computational offloading decision framework by minimizing the long-term payment of computational tasks with mixed bound constraints. In addition, redundant experiences are gotten rid of before the training of the algorithm. The most advantageous transitions in the experience pool are used for training in order to improve the learning efficiency and convergence speed of the algorithm as well as increase the accuracy of offloading decisions. The findings of our experiments indicate that the method we have presented is capable of achieving fast convergence rates while also reducing sample redundancy.**

***Index Terms*—Bound constraints, computation offloading, deep reinforcement learning (DRL), Markov decision process, multiaccess edge computing (MEC).**

I. Introduction

IN RECENT years, there has been a proliferation of a large variety of intelligent applications, which has contributed to the increased focus on multiaccess edge computing (MEC). It is an important piece of technology that can provide computational resources for resource-constrained user equipment (UE) [1], and it involves a wide variety of fields that involve offloading decision problems, such as vehicular networks and the Internet of Health Things (IoHT) [2]. In greater detail, MEC is utilized to offload the computational responsibilities of the UE to the edge node (EN), which solves the inadequacies that the devices have in terms of resource storage, computing performance, and energy efficiency [3], [4], [5]. One of the primary topics of study in the MEC is the development of methods for enabling low-latency and energy-efficient compute offloading decisions. This area of research has attracted significant interest from both academic and industrial researchers [6], [7].

Since the applications that have bound constraints, such as virtual reality (VR), IoHT, and real-time video analytics, have become incredibly common, a large amount of effort has recently been put into the design of a computation offloading policy for delay-sensitive computational tasks [8], [9], [10]. For instance, Kovacevic et al. [11] developed a formulation for the joint optimization of resource allocation for communication and computation in response to computation offloading requests that have stringent bounding restrictions. A similar design problem is tackled in [12] presenting a priority-aware computation offloading and deriving a lower latency of the average response time for all tasks. Wu et al. [13] developed a virtual queue by employing perturbed Lyapunov optimization strategies in order to convert the challenge of ensuring task deadlines into a problem of stable control for the virtual queue. Tang et al. [14] proposed a new framework in which the task cannot be finished within the bound constraints are dropped. However, the research mentioned above only considered the tasks with strict bound constraints. Some activities, such as those using multimedia, might only need to be finished before the deadlines on occasion. These tasks with a mixture of soft and hard deadlines are more in line with the reality of MEC computational offloading services, but this problem has been ignored by most work. Therefore, we intend to study the problem of computational offloading with mixed bound constraints.

Computation offloading for MEC systems requires taking into consideration the dynamics of the environment, such as time-varying network states [15]. Chen et al. [16] formulated a dynamic optimization problem as an infinite-horizon MDP model to maximize the long-term utility performance.

Manuscript received 15 February 2023; accepted 25 March 2023. Date of publication 5 April 2023; date of current version 24 August 2023. This work was supported in part by the National Key Research and Development Program of China under Grant 2022YFB4500800, and in part by the National Natural Science Foundation of China under Grant 62032013 and Grant 92267206. *(Corresponding author: Kexin Li.)*

Kexin Li is with the College of Computer Science and Engineering, Northeastern University, Shenyang 110819, China (e-mail: neulikexin@stumail.neu.edu.cn).

Xingwei Wang is with the College of Computer Science and Engineering and the State Key Laboratory of Synthetical Automation for Process Industries, Northeastern University, Shenyang 110819, China (e-mail: wangxw@mail.neu.edu.cn).

Qiang He is with the College of Medicine and Biological Information Engineering, Northeastern University, Shenyang 110169, China.

Qiang Ni is with the School of Computing and Communications, Lancaster University, LA1 4WA Lancaster, U.K.

Mingzhou Yang is with the School of Information Science and Engineering, Shenyang University of Technology, Shenyang 110178, China.

Schahram Dustdar is with the Distributed Systems Group, TU Wien, 1040 Wien, Austria.

Digital Object Identifier 10.1109/JIOT.2023.3264484

Wu et al. [17] created a technique for energy-efficient dynamic task offloading (EEDTO) by selecting the most appropriate computing location in an online manner. The implementation of the algorithm was successful in meeting the computation performance requirements because it required nothing more than the successful resolution of a deterministic challenge during each time slot. When the number of heterogeneous UEs grows, however, the dimensionality and computational complexity that were formerly a trivial barrier become a significant one.

As reinforcement learning (RL) or deep RL (DRL) is gradually applied in various industrial scenarios, mainstream researchers try to solve the computational offloading problem using the above algorithms. Scalable schemes for learning binary offloading judgments from experience are used to escape the predicament of high-dimensional states and computational complexity [18]. For instance, to demonstrate the potential of a distributed online RL system, Chen et al. provided a case study on resource orchestration in computation offloading. Shah et al. [19] expressed the multiuser computation offloading as linear programming of mixed integers problem through hierarchical deep actor–critic RL. Conventionally, DRL works on the principle of many trials and iterations of states and actions to optimize the computational offloading strategy. Nevertheless, it is challenging to converge an optimization algorithm for computational offloading tasks with bound constraints because additional features make the state space complex and high dimensional.

In this system, tasks generated by UEs need to be finished within their mixed bound constraints, making it more difficult to develop a rational and effective computational offloading mechanism. In addition, with the gradual increase of edge network users, dynamic network environments, and the explosion of state spaces, traditional RL algorithms need to be revised to handle such complex scenarios. Inspired by mentioned above, We present a new computation offloading scheme for tasks with mixed bound constraints based on experience-based RL in the MEC system. The main contributions are as follows.

1) We have developed an offloading framework in the MEC system for delay-sensitive tasks with mixed bound constraints. The goal of this framework is to minimize the long-term payment of the task. Specifically, long-term payment should include delay, energy consumption, and reasonable penalty terms to obtain the most efficient offloading strategy.
2) We devise a scheme for learning the optimal computation offloading policy with experience-based RL, which involves two parts: a) experience trimming and b) priority experience replay. It improves learning efficiency and convergence speed by removing similar transitions in the experience pool and selecting the most significant transition.
3) Numerical experiments are conducted to verify that our proposed learning scheme outperforms other baseline schemes considering the tasks with mixed bound constraints.

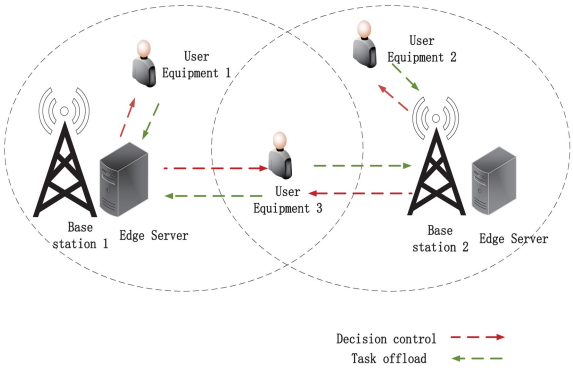


Fig. 1. Overview of MEC system.

The remainder of this article is organized as follows. In Section II, the system model is presented. The problem is transformed into an MDP-based computation offloading problem. In Section III, we propose an experience-based RL computation offloading scheme to minimize the users' payment in the MEC. Simulation results are presented in Section IV. Finally, Section V concludes this article and provides insights into possible future work.

II. System Model

In this section, we present the framework and detail the role of each module in the framework. Then, the workflow and the problem formulation are described as follows.

A. System Description

We consider a set of UEs $\mathcal{M} = \{1, 2, \ldots, m\}$ and a set of ENs $\mathcal{N} = \{1, 2, \ldots, n\}$ in this MEC system. The set of time slots indexed is defined as $\mathcal{T} = \{1, 2, \ldots, t\}$, with each decision occurring within the same time slot. An overview of this system is given in Fig. 1. In this system, each UE makes an optimal offloading method in combination with the characteristics of the task and the network conditions. After submitting the decision to the UE, the EN executes the offloading strategy.

Moreover, as a practical application, UEs may form part of an intelligent device or Internet of Things (IoT) system application. In this article, we focus only on the computationally intensive tasks of UEs and assume that they are generated on UEs at the rate of Poisson distribution λ. The connectivity between each UE and EN is assumed to be wireless fidelity with a finite computation capability. It depends on the offloading decision whether these tasks should be processed locally or offloaded to an EN n. Each EN consists of a base station (BS) and an edge server (ES) as shown in Fig. 1. The BS provides communication services, and tasks are uploaded to the ES over a wireless channel. The ES offers computational resources to the UE, assuming that the computational capacity of the ES far exceeds that of the UE. At the beginning of each time slot, a new task arrives at the UE m. The control center deployed on the edge service obtains global information, such as data size and computational capacity of UE m and EN n. Then, it makes a computational offloading decision based on the above information.

B. Task Model

Without loss of generality, we assume that each UE has only one computational task to process at a time and that tasks cannot be further partitioned. Therefore, the computation offloading decision is dichotomous, and tasks can only be executed locally or on EN. To make the tasks more visible and intuitive, let C_m and f_m (cycles per second) be the data size of the task and the computation rate of UE m, respectively. Specifically, we consider the task of UE m with mixed deadlines bound$_m = \{t_m^s, t_m^h\}$, where t_m^s and t_m^h mean the task of UE m's soft bounds and hard bounds, respectively.

C. Computation Model

Depending on the current state, the computation offloading policy of UE m only in two cases, "local" or "offload." The two computational models are described in detail as follows.

1) Processing Locally: At time t, the task of UE m's execution delay in this case is defined as $D_m^l(t)$. And we assume that each UE's computational capabilities can be different, defined as f_m. Then, the execution delay in this case can be expressed as

$$D_m^l(t) = \frac{C_m}{f_m}. \tag{1}$$

Another point to consider is that according to the circuit theory [20], the amount of energy consumed by the CPU is directly proportional to the circuit supply voltage performance. Additionally, when the processor works at low voltage limits, the circuit-supplied voltage is linearly proportional to the computing speed of the processor. In consequence, it is possible to express the energy consumption per CPU cycle as follows:

$$E_m^l(t) = \epsilon C_m \left(D_m^l\right)^2 \tag{2}$$

where ϵ is defined as the effective switching capacitance, which depends on the chip structure according to [21].

Based on (1) and (2), the payment $p_m^l(t)$ for the computing task of UE m locally is calculated as

$$p_m^l(t) = \sum_{t=1}^{\mathcal{T}} \omega_1 D_m^l(t) + \omega_2 E_m^l(t) \tag{3}$$

where $0 < \omega_1 < 1$ and $0 < \omega_2 < 1$ are set as the weights of probability.

2) Processing via Offloading: If UE m offloads the task to EN n, the delay is composed of three components: 1) the transfer delay for uploading the task to EN n; 2) the execution delay; and 3) the transfer delay for returning the execution result to UE m. These three components will be explained in detail below.

1) *Task Uploading:* We assume that the wireless channel is different between time slots as it has independent and identically distributed block fading same as [22]. We define G_{mn}^{up} as the small range of channel power gain from UE m to EN n. About all, we can calculate the corresponding channel power gain $h_{mn}^{t,\text{up}}$ based on

$$h_{mn}^{t,\text{up}} = G_{mn}^{\text{up}} \varphi (d_{\text{ref}}/d_{mn}) \tag{4}$$

where φ, d_{ref}, and d_{mn} are defined as the passing loss constant, the reference distance, and the actual distance from UE m to EN n, respectively. We set $b_{mn}^{t,\text{up}} \in [B_{\min}, B_{\max}]$ as the network bandwidth, where $B_{\min}$ and $B_{\max}$ are the minimum and maximum network bandwidths, respectively. Based on the above description, the communication rate can be expressed as follows:

$$r_{mn}^{\text{up}} = b_{mn}^{t,\text{up}} \log_2 \left(1 + \frac{h_{mn}^{t,\text{up}} \cdot P_{mn}}{IN\sigma^2}\right) \tag{5}$$

where P_{mn} is the transport power of communication, σ^2 is the unit distance path loss, and IN is set as the signal-to-interference-plus-noise ratio.

If UE m offloads its computational task to EN n at time t, the task completion delay shall include the data transfer delay to the EN. We define C_m as the task size allocation from UE m to EN n. Thus, the transfer delay from UE m to EN n at time t can be expressed as follows:

$$D_{mn}^{\text{up}}(t) = \frac{C_m}{r_{mn}^{\text{up}}}. \tag{6}$$

Furthermore, the transmission energy consumption at EN n is given by

$$E_{mn}^{\text{up}}(t) = P_{mn} D_{mn}^{\text{up}}(t). \tag{7}$$

2) *Task Execution:* EN n's computing speed can be calculated as f_n. Consequently, the computation delay in this case is

$$D_{mn}^{ex}(t) = \frac{C_m}{f_n}. \tag{8}$$

Suppose that UE m keeps an idle state while the EN executes the task, and the energy consumption of UE m in the idle state needs to be considered, which is defined as P_m^{idle}. The energy consumption can be obtained as

$$E_{mn}^{\text{idle}}(t) = \frac{P_m^{\text{idle}} \cdot C_m}{f_n}. \tag{9}$$

3) *Results Downloading:* Let consider the symmetric channel, and the transmission rate of the wireless downlink from the EN n to UE m is calculated as $r_{mn}^{\text{down}} = b_{mn}^{t,\text{down}} \cdot \log_2(1 + [h_{mn}^{t,\text{down}} \cdot P_{mn}]/[IN \cdot \sigma^2])$. $h_{mn}^{t,\text{down}}$ denotes the received channel power gain from EN n to UE m, I is the signal-to-interference-plus-noise ratio set. The data size of the task of UE m execution result is represented as z_m. As a result, the expression for the transfer delay for downloading execution results from EN n is

$$D_{mn}^{\text{down}}(t) = \frac{z_m}{r_{mn}^{\text{down}}}. \tag{10}$$

In addition, define P_{mn}^{down} as the energy consumption of UE m to download the execution results from EN n. The energy consumption of UE m during the downloading of results can be obtained by the following equation:

$$E_{mn}^{\text{down}}(t) = \frac{P_{mn}^{\text{down}} \cdot z_m}{r_{mn}^{\text{down}}}. \tag{11}$$

Without loss of generality, the total completion time for EN and energy consumption of UE m can be calculated with the following formulas:

$$D^o_{mn}(t) = D^{\text{up}}_{mn}(t) + D^{ex}_{mn}(t) + D^{\text{down}}_{mn}(t) \tag{12}$$

$$E^o_{mn}(t) = E^{\text{up}}_{mn}(t) + E^{\text{idle}}_{mn}(t) + E^{\text{down}}_{mn}(t). \tag{13}$$

Based on (12) and (13), the payment $p^o_m(t)$ for the offloading task of UE m is calculated as

$$p^o_m(t) = \sum_{t=1}^{\mathcal{T}} \omega_1 D^o_{mn}(t) + \omega_2 E^o_{mn}(t) \tag{14}$$

where $0 < \omega_1 < 1$ and $0 < \omega_2 < 1$ are set as the weights of probability.

D. Problem Formulation

Since the MEC problem is a temporal decision problem, existing work has demonstrated that the computational offloading problem can be solved as an MDP problem [23]. Specifically, a decision-capable agent can observe the environment state $\mathcal{S}$, i.e., network information, and perform an action $\mathcal{A}$ for a task based on the state information, i.e., whether the task is offloaded or not, which determines the next global state. The task of UE m is rewarded with $\mathcal{R}$, which is used to determine the excellence of the action. Typically, an intelligence maximizes its expected reward by optimizing the policy mapping from states to actions; in the system we consider, our goal is to minimize long-term payments. Next, we introduce the three essential elements in this system: 1) state; 2) action; and 3) reward. Then, we construct the payment minimization problem for the system.

State: In this system, before the start of each time slot, the UE m obtains the state information, including the data size of the task, the task boundary constraints, and the computational power of the EN. The state can be represented as a triple $s_t = \{C_m, \text{bound}_m, f_n\}$. Let $\mathcal{S}$ denote the discrete and finite state space of each UE, i.e., $\mathcal{S} = C_m \times \text{bound}_m \times \{f_1, f_2, \ldots, f_N\}^N$, where C_m is the size of the data for task m at time t, bound_m is the constraint of the task, and f_n is the computational speed (number of cycles per second) of EN n.

Action: To represent the manner in which the task is executed, we define the action as $\mathcal{A}_t = a_t = \{x_m(t)\}$, $m \in \{1, \ldots, \mathcal{M}\}$, $x_m(t) \in \{0, 1\}$. $x_m(t) = 0$ means that the task of UE m will be assigned to the EN; and vice versa.

Reward: If a task has been processed, the task latency is the time between task arrival and task completion, and the energy consumption is the total payment of computing energy and communication payment. We set a binary variable $x_m(t) = \{0, 1\}$ to denote the unique index of the task at the beginning of time slot $t \in \mathcal{T}$. If the task will be processed locally, $x_m(t) = 1$, and vice versa. Based on (3) and (14), we defined $P_{\text{total}}(t)$ as the total task payment of UE m, i.e.,

$$P_{\text{total}}(t) = \sum_{t \in \mathcal{T}, m \in \mathcal{M}} x_m(t) p^l_m(t) + |1 - x_m(t)| p^o_m(t). \tag{15}$$

The reward function of a model is usually related to the system's objective. In this situation, the offloading framework makes decisions by minimizing the system's payment. Therefore, we find a utility function $u(s_t, a_t)$ that can be used to quantify the reward calculated for the tasks of the UE in the current model. In this case, we define the service reward in this system as $u(s_t, a_t)$, which can be defined as follows:

$$u(s_t, a_t) = \arg\min_{s_t, a_t} P_{\text{total}}(t). \tag{16}$$

We design an incentive mechanism to encourage them to prioritize important tasks. Specifically, if the task of UE m is completed before its soft bounds t^s_m, the agent receives reward $\mathcal{R}(s_t, a_t)$. If the task of UE m is completed before its hard bound t^h_m but exceeds its soft bounds t^s_m, the agent receives penalty $\varsigma^{\text{soft}}_m$. Otherwise, the agent receives penalty $\varsigma^{\text{hard}}_m$. Generally, the failure to complete a task with a hard deadline incurs a higher penalty than that of a task with a soft deadline, i.e., $\varsigma^{\text{soft}}_m \le \varsigma^{\text{hard}}_m$.

Then, the long-term reward can be formulated as

$$\mathcal{R}(s_t, a_t) = \sum_{m \in M} \left\{ \frac{1 - f(g_m)}{2} \cdot u(s_t, a_t) - \left[1 - \frac{1 - f(g_m)}{2} \right] \cdot \varsigma_m \right\} \tag{17}$$

where function $f(x)$ used in (17) is defined as $f(x) = A \cdot \mathbf{1}(x)$, A is a unit step function, and $g(m) = x_m(t) D^l_m + |1 - x_m(t)| D^o_m - t^d_m$. Once the task of UE m is completed before its soft bounds, $f(x) = -1$ and $([1 - f(g_m)]/2) = 1$. Therefore, the reward will be produced. On the contrary, once the mixed bounds of the task are missed, $f(x) = 1$ and $([1 - f(g_m)]/2) = 0$, the penalty $\varsigma_m = \{\varsigma^{\text{soft}}_m, \varsigma^{\text{hard}}_m\}$ is incurred.

In addition, the policy π_m of UE m is a mapping from its states s_t to its action a_t. The objective of this system is to search the policy strategy π_m for each UE m and minimize the agent's reward. Based on (15)–(17), our optimization objective is defined as minimizing the expectation of reward

$$\pi^*_m = \arg\min_{\pi_m} \mathbb{E}\left[\frac{1}{\mathcal{T}} \sum_{t=1}^{\mathcal{T}} \gamma^{t-1} \mathcal{R}(s_t, a_t) | \pi_m \right] \tag{18}$$

where $\gamma^{t-1} \in (0, 1]$ denotes the discount factor that reflects the discount reward of the future. $\mathbb{E}[\cdot]$ is concerned with the system parameters, i.e., bound constraints, and processing requirements of all UEs, and further influences the offloading strategy.

III. Deep Reinforcement-Learning-Based Approach

In this section, we propose an experience-based deep RL (EBRL) offloading algorithm that enables dynamic offloading decision making of each UE. Like the normal DRL algorithm for MEC offloading problems, i.e., deep Q-learning, the proposed algorithm decides by memorizing and reusing past information by experience replay. Meanwhile, the proposed algorithm can improve the convergence rate and the impossible issue of high dimensionality in state space.

In the EBRL algorithm, each UE makes an offloading strategy by learning a mapping from each state–action pair to a Q-value, reflecting the policy's expected long-term payment. The offloading policy depends on the result of the neural network, which obtains the result depending on the samples

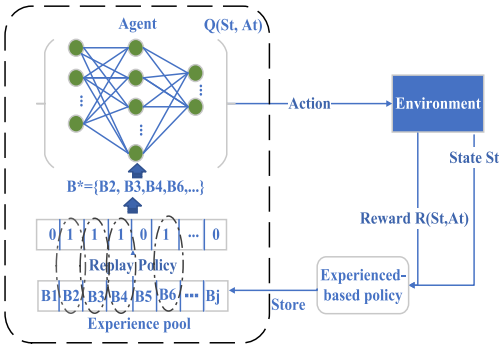


Fig. 2. Illustration of experienced-based RL.

from the experience pool. In the following, we present the EBRL algorithm in detail.

A. Experience Replay Method

The objective of the experience replay method is to consider two questions, *Which* information is worthy of being put into the experience pool, and *How* to choose the most significant experience to put into the neural network. We propose an experience replay method, including an *Experience Trimming method* and a *Priority Experience Replay method* to solve the above two questions.

Fig. 2 shows the details of the experience replay method for UE $m \in \mathcal{M}$. Each UE can be seen as an agent that takes an action $a_t \in \mathcal{A}$ in state $s_t \in \mathcal{S}$, receives a reward r_t, and observes the next state s_{t+1}. The quadruple (s_t, a_t, r_t, s_{t+1}) is stored in the experience pool at each time slot and also named transition. Before the transition is stored in the experience pool, the most valuable transition needs to be selected by an experienced trimming method. Afterward, a batch of transitions drawn from the experience pool is trained by the priority experience replay method to update the agent information and generate the offloading decision.

1) Experience Trimming Method: Experience replay has played an important part in remembering and learning from experiences from the past of DRL. There have been several proposed studies into how to experience replay can influence the DRL algorithms [24]. However, it is nontrivial to model an appropriate replay policy for MEC offloading problems for several challenges. On the one hand, the transitions in the experience pool are quite noisy due to the dynamic network environment. On the other hand, there are a lot of repeating transitions in the experience pool. To the best of our knowledge, there are few types of research on which samples can be stored in the experience pool. Therefore, it is challenging to effectively and efficiently learn the replay policy.

The results of the neural network directly affect the computational offloading decision. During training, the neural network takes as input historical transitions and outputs an approximate Q value $Q(s_t, a_t)$. Then, the state s_t interacts with the environment, performs the action a_t according to the policy $pi(s_t, a_t)$, and gets the corresponding immediate reward r_t based on the state action pair. The transition information is stored as (s_t, a_t, r_t, s_{t+1}) $(s_t \in \mathcal{S}, a_t \in \mathcal{A}, r_t \in \mathcal{R}$, and $s_{t+1} \in \mathcal{S})$ in the experience pool as memory. Let $\mathcal{B}_j$ be a transition in experience pool $\mathcal{B}$, J is the size of the experience pool, where $j \in J$ denotes the index. In this article, the experiential log of each time slot is denoted as $\mathcal{B} = <\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_t, \mathcal{S}_{t+1}>$.

Algorithm 1: Experience Trimming Method

Input: $\mathcal{B}_j = \{(s_t, a_t, r_t, s_{t+1})_j, j \in \mathcal{J}\}$,
$\mathcal{B} = (s_{t^*}, a_{t^*}, r_{t^*}, s_{t^*+1}), t^* \in (1, 2, \ldots t-1)$
Output: New experience pool $\mathcal{B}$
Initialize similarity ***Similarity level***
for *each replay updating step* **do**
  Calculate similarity based on (19)
  Update the experience pool $\mathcal{B}$
end

In order to simplify the memory structure and improve the effectiveness of the information, we developed a criterion to measure the importance of the current incoming information. If these experience transitions had similar characteristics, we combined some experience transitions within one piece of information in the experience pool. It simplifies the experience pool structure by combining similar historical transitions into one transition if they have similar characteristics. Furthermore, we define a similarity function that measures the similarity between the transition and historical experience of the learning agent by evaluating two metrics.

1) The users' information, which refers to the size of tasks, device patterns, etc.
2) The subchannel information, which refers to the utilized energy, transfer rate, etc.

Jaccard similarity coefficient is a statistic used for gauging the similarity and diversity between finite sample sets [25]. Define A to be the experience transitions already stored in the experience pool and B to be the experience transitions that are currently about to arrive. Before the experience transition B is stored in the experience pool, the two are compared. Assuming that both experience transitions satisfy the condition $J(A, B) < \vartheta$, the best reward value experience transition in the experience pool will be stored, and the other one will be dropped until the experience pool is full. The similarity level between two transitions is given by

$$J(A, B) = \frac{|A| \cap |B|}{|A| \cup |B|} \tag{19}$$

where $J(\cdot)$ denotes the well-known Jaccard similarity coefficient, manifolding the similarity between set A and set B. $\vartheta = 0.5$ is the similarity level in the simulation. When the experience pool is full, the last used historical experience transition will be dropped and the new coming experience transition will be stored. The Experience Trimming method is summarized in Algorithm 1.

2) Priority Experience Replay: It is well known that experience replay makes DRL more efficient by remembering and reusing experiences from the past [24]. Most commonly for MEC computational offloading problems, the experience replay is sampled uniformly randomly from the experience pool when training the neural networks [26].

Algorithm 2: Priority Experience Replay

Input: experience pool $\mathcal{B}$, batch size $\mathcal{I}$
Output: Sampled batch transition $\mathcal{B}^*$

Initialize batch transition $\mathcal{B}^* = \varnothing$
for *i=1 to* $\mathcal{I}$ **do**
  Store (s_t, a_t, r_t, s_{t+1}) in $\mathcal{B}^*$ with maximal priority based on (21)
  Update the priority $\mathcal{P}_j$ according to p_j
  Update the batch transition based on (23)
end
Sample a subset $\mathcal{B}^*$ from $\mathcal{B}$

Although the random sampling policy is an easy default, the performance of the DRL algorithm can be improved by using strategies to choose the experience samples used for training [27], [28]. Therefore, unlike the previous design of randomly sampling replay in the DRL applied for MEC offloading system, we adopt a priority experience replay method to sample subsets of transitions for efficient replaying.

The UE is assumed to be equipped with an experience pool of finite size to store the experience transition which can be represented as $\mathcal{B}_j = \{\{s_t, a_t, r_t, s_{t+1}\}_j, j \in \mathcal{J}, t \in \mathcal{T}\}$. According to the experience replay technique [22], the UE then samples a batch transition $\mathcal{B}^*$ from the experience pool according to the reply policy at each decision epoch to train the neural network. In the learning phase, the experience pool stores the $\mathcal{B}_j$ continuously. In the training phase, for each training interval, the agent selects a batch $\mathcal{B}^*$ from experience pool $\mathcal{B}$ and trains it. At each training epoch, $\mathcal{B}^*$ can be represented as $\mathcal{B}_i^* = \{\{s_t, a_t, r_t, s_{t+1}\}, i \in 1, 2, \ldots, \mathcal{I}, t \in \{1, 2, \ldots, \mathcal{T}\}\}$, $\mathcal{I}$ is the size of the batch transition, where $i \in \mathcal{I}$ denotes the index. The algorithm of experience replay is defined as Algorithm 2.

In our preliminary experiments, we find that the temporal-different error can measure the significance of the sample. A greater absolute value of the temporal-different error means a greater loss of the Q network during training. This suggests that samples with more significant temporal-different errors contain more information and increase the accuracy row of the Q network. Therefore, samples with larger temporal-different errors will have a higher probability of being a subset than other experiences. Further, we develop sampling priorities for samples determined by the distribution of sample temporal-different errors. The temporal-different error of the experience transition j can be expressed as

$$\delta_j = \mathcal{R}_j + \gamma_j Q_{\text{target}}\big(\mathcal{S}_j, \text{argmax}_a Q(\mathcal{S}_j, a)\big) - Q\big(\mathcal{S}_{j-1}, \mathcal{A}_{j-1}\big). \tag{20}$$

However, it is infeasible to sweep over the entire experience pool because the experience pool size is usually large. In this priority experience replay, the replay policy is described as a priority score function $\phi(\cdot)$, in which a higher value indicates a higher probability of the transition $\mathcal{B}_j$ be selected in the sampled subset. In this work, the chosen probability of the batch transition can be expressed as

$$p_j = \frac{\delta_j}{\sum_{\mathcal{J}} \delta_{\mathcal{J}}}. \tag{21}$$

We describe the priority score function as

$$\mathcal{P}_j = \big\{\phi\big(p_j|\mathcal{B}_j\big)\big\} \in \mathbb{R}^N \tag{22}$$

where $\phi \in (0, 1)$ denotes a function approximation which is a deep neural network. Given the priority score $\mathcal{P}_j$, we then sample batch transition $\mathcal{B}^*$ according to

$$\mathcal{B}^* = \big\{\mathcal{B}_j|\mathcal{B}_j \in \mathcal{B} \wedge \mathcal{P}_j = 1\big\}. \tag{23}$$

Algorithm 3: EBRL Algorithm to Solve the Offloading Decision Problem

Input: $Q(s, a), \forall s_t \in \mathcal{S}, a_t \in \mathcal{A}$, and $Q(terminal - state, \cdot) = 0$
Output: Offloading action a_t

Initialize s_t, max similarity $= \vartheta$;
for *time step* $= 1 : \mathcal{T}$ **do**
  Repeat (for each step of episode):
  Choose Action a_t and state s_t using policy derived from Q function (e.g.,ϵ-greedy);
  Take action $\mathcal{A}$, reward $\mathcal{R}$, and $\mathcal{S}_t$
  $Q(S_t, a) \leftarrow$ $Q(S_t, a)+\alpha[R + \gamma max_\alpha Q(S_{t+1}, a) - Q(S_t, a)]$
  $S_t \leftarrow S_{t+1}$
  Find the historical learned action $a_{historical}$;
  for $m = 1 : \mathcal{J}$ **do**
    Update similarity using (19)
    Store (s_t, a_t, r, s_{t+1}), in experience pool $\mathcal{B}$ using Algorithm 1
  end
  for *n=1* $: \mathcal{I}$ **do**
    Sample experience B_s with probability P_j in (21)
    Update the priority $\mathcal{P}_j$ according p_j
  end
  Update the θ_j and $\theta_{target,j}$ using (26) (27)
  Update the current action $a_{current}$ using (28)
  update offloading action a_t using (29)
end

B. Experience-Based Reinforcement Learning

The EBRL algorithm makes the offloading policy to indicate the task to be executed at UE $m \in \mathcal{M}$ or EN $n \in \mathcal{N}$. The detail of this method is given in Algorithm 3. The key idea of the algorithm is to use the experience of UE to train the neural network and obtain the mapping from the state–action pair (s_t, a_t) to the Q-value. Therefore, on this basis, the UE can choose the action that minimizes the Q-value in the observed state and minimizes its expected long-term reward r_t.

Hence, the EBRL $Q(s_t, a_t; \theta)$ approximates the optimal state–action function $Q(s_t, a_t)$ can be given as

$$Q(s_t, a_t; \theta) = \sum_j Q\big(s_t, a_t; \theta_j\big) \tag{24}$$

TABLE I
DEFAULT SIMULATION PARAMETERS

Parameter	Description	value
M	Number of UE	(100,1000)
N	Number of EN	10
$b_{mn}^{t,up}, b_{mn}^{t,down}$	Bandwidth	[0.5,200]MHz
B_{min}	Minimize bandwidth	0.5MHz [30]
B_{max}	Maximize bandwidth	200MHz [30]
P_{mn}	Transmit power	25dBm
$G_{mn}^{t,up}, G_{mn}^{t,down}$	Fading channel power gain	-5dB
$h_{mn}^{t,up}, h_{mn}^{t,down}$	Channel gain	-25dB
ϵ	Effective switched capacitance	0.5 [31]
d_{mn}	Distance between UE and EN	35m [30]
k	Number of experience transition	64 [30]
σ^2	Pass loss	4 [30]
d_{ref}	Reference distance	50m
f_m	UE computing capacity	10GHz [32]
f_n	EN computing capacity	50GHz [32]
τ	temperature	0.5 [33]

where $\theta = (\theta_j)$ is a collection of parameters associated with the DNN. At each epoch, the agent maintains a DNN and a target DNN, which are $Q(s_t, a_t; \theta)$ and $Q(s_t, a_t; \theta_{\text{target}})$, respectively. Our proposed experience replay strategy is executed in which the intelligence is trained by drawing minibatch $\mathcal{B}^*$ from the experience pool $\mathcal{B}$. Then, the parameters θ_j are updated according to the loss function using gradient descent. Typically, the loss function can be expressed as the mean square measure of the Bellman equation error [29]. Therefore, the loss function $L_{(\text{EBRL})}(\theta_j)$ is calculated by the equation

$$L_{(\text{EBRL})}(\theta_j) = E[(R(s_t, a_t) + \gamma Q(s_{t+1}, \text{argmax}Q(s_{t+1} a_{t+1}; \theta_{\text{target},j}))) - Q(s_t, a_t)]. \quad (25)$$

Next, the parameters of theta and theta$_{\text{target}}$ are updated, where *mu* is the learning rate of the gradient decay algorithm and $nu/in(0, 1)$ is the weight parameter

$$\theta_j \leftarrow \theta_j + \mu \cdot \nabla L_{(\text{EBRL})}(\theta_j) \quad (26)$$

$$\theta_{\text{target},j} \leftarrow \nu\theta_j + (1 - \nu)\theta_{\text{target},j}. \quad (27)$$

According to the policy $pi(s_t, a_t)$, the agent chooses the action a_t under the state s_t, where $pi(s_t, a_t)$ is defined as

$$\pi(s_t, a_t) = \frac{\exp(Q(s_t, a_t; \theta)/\tau)}{\exp(Q(s_t, a_t'; \theta)/\tau)} \quad (28)$$

where τ is temperature, the smaller the τ is, the more likely action a_t is to be selected.

To further enhance the exploration performance of the algorithm, we consider historical action $a_{\text{historical}}$ (the action stored in the experience pool) and current action a_{current} (the agent chooses the action for the current task) in the action selection process, and the action choosing formulation can be expressed as

$$a = o * a_{\text{historical}} + (1 - o)a_{\text{current}} \quad (29)$$

where $o \in [0, 1]$ denotes the transfer rate, and the value will decrease with the stage step increased information. The proposed EBRL algorithm for offloading is provided in Algorithm 3.

C. Computational Complexity Analysis

The proposed EBRL approach have defined $\mathcal{T}$ time steps, i.e., $\mathcal{T} = \{1, 2, \ldots, t\}$, the computational complexity can be simplify expressed as $O(\mathcal{T}^2)$ [34]. However, the complexity of the algorithm should not be neglected to many important parameters, i.e., the number of UE, and the batch size of the experience pool. Therefore, we analyze the computational complexity of two modules (Experience Trimming method and Priority Experience Replay method) and analyze their computational complexity in the following.

Define the system with UE P and an experience pool of size I, for the Experience Trimming method. According to [35], the complexity of the Experience Trimming method is $\mathcal{O}(I)$. Further, for the Priority Experience Replay method, Q is defined to denote the dimension size of the state space, U denotes the size of the minibatch, and V denotes the maximum plot during training, and its complexity is $\mathcal{O}(P^2 \cdot I)$ according to Algorithm 2. In addition, the computational complexity of one experience training is defined as $\mathcal{O}(L)$, where L is the number of multiplication operations in the neural network, and the computational complexity of EBRL is $\mathcal{O}(LI^2P^2QV)$ according to [36].

IV. SIMULATION RESULTS AND DISCUSSION

A. Simulation Settings

Throughout the experiments, we assume that UEs are randomly distributed within an area of 350 m × 350 m. Additionally, the reference distance, channel bandwidths between the UE and the ENs, and the transmit power from UEs to ENs are 50 m, 6 MHz, and 25 dBm, respectively [30].

For the task execution, the task bound constraints to follow the uniform Quality of Service (QoS) between [5, 30] s, while the gap from the soft bounds to hard bounds is [0, 20] s according to [31]. In addition, the task data sizes follow the uniform distribution on [100, 1000] kB. The arrival rate of the task is 0.5 and the pass-loss constant is 4. The fading channel power gain and channel gain are –5 and –25 dB, respectively.

We assume that the basic parameters for the proposed EBRL method are a time slot of $1e^{-2}$, the action slot of $5e^{-2}$, and the time total of 60 000 s. Without losing generality, the initial values of ω_1, ω_2, and o for the EBRL parameters are 0.5, 0.5, and 0.8, respectively [33]. The target network parameter is 0.8. The other default simulation settings are listed in Table I.

B. Convergence Analysis

To reflect the change in the convergence of our proposed algorithm, we compare the number of iterations required to converge for different learning rates for the EBRL and deep deterministic policy gradient (DDPG) algorithms. Fig. 3 shows that as the learning rate increases, the number of iterations required to converge decreases accordingly for both methods. This is because the agent learns more information from the new transitions as the learning rate increases. Further, Fig. 3 also shows that the EBRL method reaches the convergence state faster than DDPG. This is because EBRL focuses on meaningful transition experiences, and compared to DDPG for

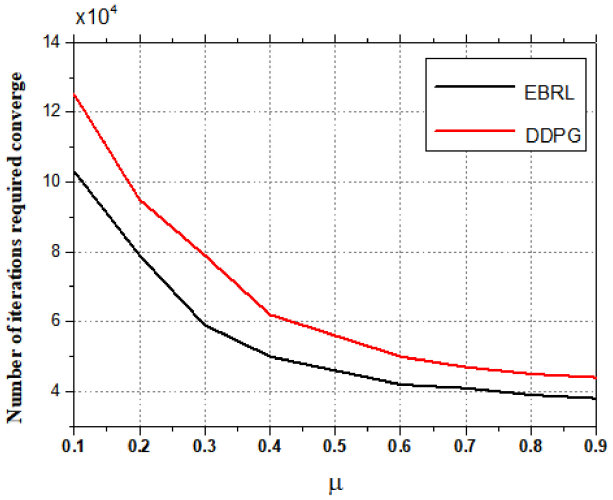


Fig. 3. Number of iterations required to converge as the learning rate μ varies.

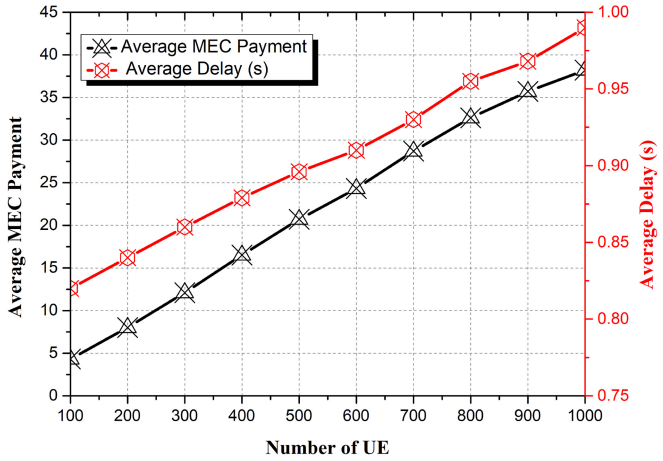


Fig. 4. Average MEC payments as the number of UE varies.

random sampling, EBRL can control for worthless transitions from incorrect training steps in future updates. Thus, the agent can learn vital information to speed up convergence.

C. System Performance Under Different Features

A larger number of UE and the task's Poisson distribution will cause invalid training data to increase system delay. Therefore, we explore the number of UE effects on the average system payment and delay. Fig. 4 shows how this system performs with different numbers of UEs based on average MEC payments and average delay. From 100 to 1000 UEs, the average MEC payment and the average delay increase by 14.43% and 69.76%, respectively. The local computing resources cannot keep up with the computing needs of the users as the number of UEs increases. Therefore, a large number of tasks need to be transferred to the EN.

Fig. 5 shows the changes in the average delay and average MEC payment as the Poisson distribution of the task changes. In Fig. 5, these curves show that as the tasks' Poisson distribution rises, the delay and payment of tasks appear to increase by

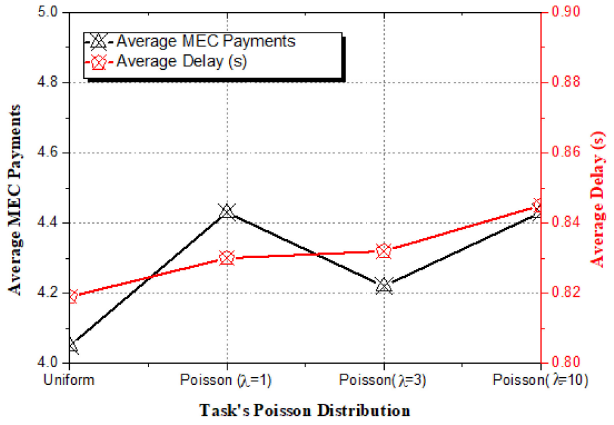


Fig. 5. Average MEC payments as the Poisson distribution of tasks varies.

varying degrees, which validates our previous theoretical analysis. This is because as the Poisson distribution of tasks rises, the computational tasks increase, leading to fewer local computational resources being allocated to each task. Furthermore, we can observe that the increase in the Poisson distribution of tasks leads to a more complex dynamic environment. When the number of users grows, it is necessary to improve the local and edge computing resources allocated to each user to minimize the processing latency and improve the computational offloading efficiency.

D. System Performance Under Different Methods

In this section, we evaluate the proposed offloading policy by comparing it with several benchmarks [DDPG [32], soft actor–critic (SAC) [37], EDGE (offload all tasks to EN), and LOCAL (execute all tasks locally)]. The DDPG and SAC methods are widely used in computation offloading decision problems, EDGE and LOCAL are currently two offloading methods widely used as benchmark comparison methods.

In Fig. 6, we evaluate the system performance under different task arrival rates and algorithm settings. With increasing task arrival rates, the average energy consumption decreases significantly for all four methods in Fig. 6(a). In general, this is due to the fact that rising task arrival rates require each UE to be assigned more computational resources. Compared to the other benchmark methods, which increase by at least 33.32% over a growth rate of 1.0–3.5 Mb/s, EBRL grows by only 23.07% over a growth rate of 1.0–3.5 Mb/s. The proposed method focuses on introducing the experience replay mechanism. This makes it possible to generate more efficient decision results based on historical information when high task arrival rates occur without having to repeat the training process. Additionally, it is worth mentioning that as the task arrival rate increases, our algorithm converges to a stable result more quickly than the other methods when the task arrival rate increases. This is because compared to the same frequency of training history data for other methods, EBRL selects more important experiences to participate in the training improves the accuracy of the results. For the same reason, the proposed algorithm also outperforms when the number of

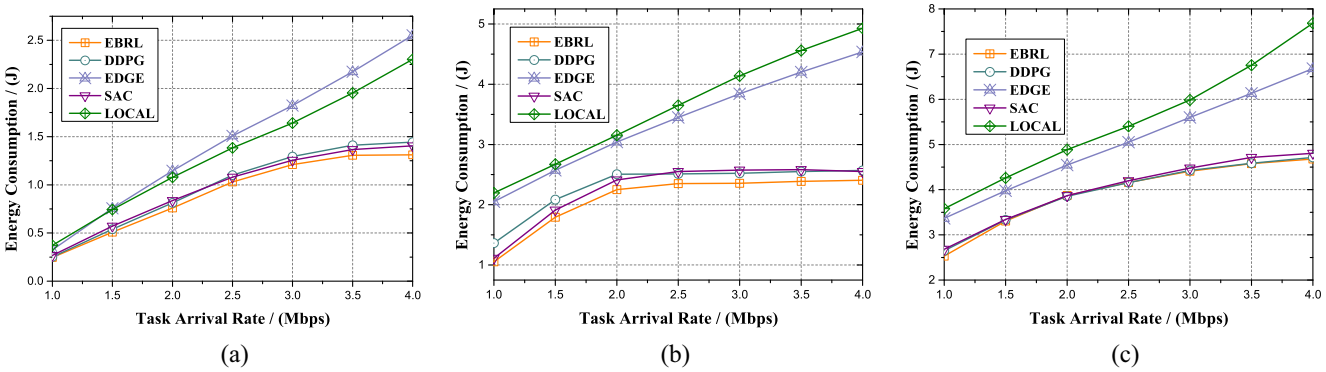


Fig. 6. Performance under different task arrival rate versus energy consumption: (a) number of UEs is 10; (b) number of UEs is 50; and (c) number of UEs is 100.

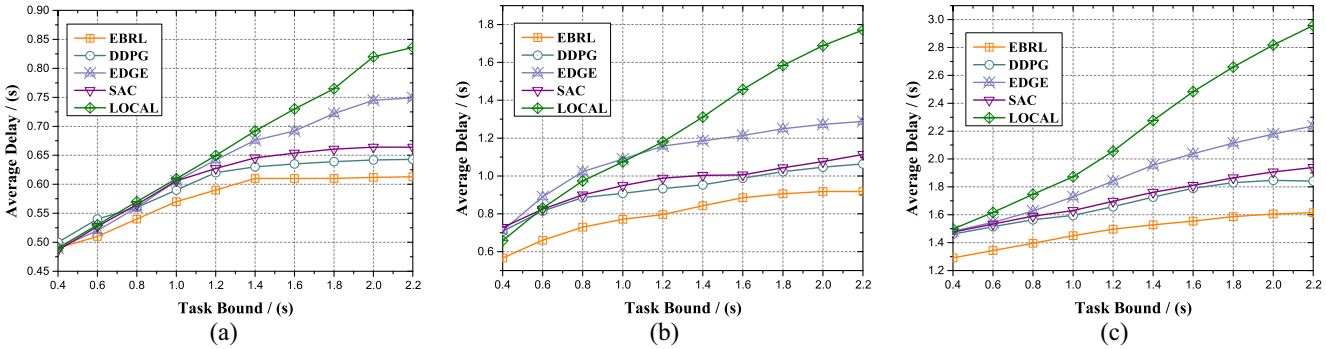


Fig. 7. Performance under different task bound versus average delay: (a) number of UEs is 10; (b) number of UEs is 50; and (c) number of UEs is 100.

tasks increases, as shown in Fig. 6(b) (number of UEs is 50) and Fig. 6(c) (number of UEs is 100).

In Fig. 7, we evaluate the performance comparisons of the algorithms under different task constraints and parameter settings. Fig. 7(a) compares the average delay obtained by the proposed algorithm with the benchmark method when the number of UEs is 10. As shown in Fig. 7(a), our approach only slightly impacts task delay as the task bound constraints rise. On the other hand, EDGE and LOCAL are significantly different in terms of the shift in task bound constraints. This is mainly because our method can complete the task briefly since the experience reply mechanism. It should be noted that when the task bound constraint was increased from 0.4 to 2.2 s, our average latency increased by 19.67%, while those of the benchmark methods increased by 26.15%–41.67%. As a consequence of the same reason, the proposed algorithm also performs well when the number of tasks increases, and this is illustrated in Fig. 7(b) in which a count of 50 UEs is shown, while Fig. 7(c) shows a count of 100 UEs.

V. Conclusion

In this article, we investigate RL-based computation offloading in multiuser MEC systems. To guarantee users' QoS, we construct a computational offloading framework by minimizing the payment of users with mixed bound constraints. To address the above issues, we propose an experience-based RL approach to solve the payment minimization problem efficiently by introducing two experience replay mechanisms. Numerical results verify the effectiveness of the presented EBRL approach and achieve better performance than other algorithms in MEC.

This work can be extended by considering the mobility of the device in the computation offloading problem. In order to modify the actual MEC system, we will also take into account the costs of computational services for UEs.

References

[1] H. Djigal, J. Xu, L. Liu, and Y. Zhang, "Machine and deep learning for resource allocation in multi-access edge computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 24, no. 4, pp. 2449–2494, 4th Quart., 2022.

[2] J. Xu, B. Ai, L. Chen, Y. Cui, and N. Wang, "Deep reinforcement learning for computation and communication resource allocation in multiaccess MEC assisted railway IoT networks," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 12, pp. 23797–23808, Dec. 2022.

[3] J. Lu et al., "A multi-task oriented framework for mobile computation offloading," *IEEE Trans. Cloud Comput.*, vol. 10, no. 1, pp. 187–201, Jan./Mar. 2022.

[4] M. Chen, S. Guo, K. Liu, X. Liao, and B. Xiao, "Robust computation offloading and resource scheduling in cloudlet-based mobile cloud computing," *IEEE Trans. Mobile Comput.*, vol. 20, no. 5, pp. 2025–2040, May 2021.

[5] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Trans. Mobile Comput.*, vol. 17, no. 2, pp. 461–474, Feb. 2018.

[6] W. Fang, S. Ding, Y. Li, W. Zhou, and N. Xiong, "OKRA: Optimal task and resource allocation for energy minimization in mobile edge computing systems," *Wireless Netw.*, vol. 25, no. 5, pp. 2851–2867, 2019.

[7] L. Lei, H. Xu, X. Xiong, K. Zheng, W. Xiang, and X. Wang, "Multiuser resource control with deep reinforcement learning in IoT edge computing," *IEEE Internet Things J.*, vol. 6, no. 6, pp. 10119–10133, Dec. 2019.

[8] Z. Wang et al., "Energy-aware and URLLC-aware task offloading for Internet of Health Things," in *Proc. IEEE Global Commun. Conf.*, 2020, pp. 1–6.
[9] W. Fan, L. Yao, J. Han, F. Wu, and Y. Liu, "Game-based multitype task offloading among mobile-edge-computing-enabled base stations," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17691–17704, Dec. 2021.
[10] 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.
[11] I. Kovacevic, E. Harjula, S. Glisic, B. Lorenzo, and M. Ylianttila, "Cloud and edge computation offloading for latency limited services," *IEEE Access*, vol. 9, pp. 55764–55776, 2021.
[12] M. Mukherjee et al., "Delay-sensitive and priority-aware task offloading for edge computing-assisted healthcare services," in *Proc. IEEE Global Commun. Conf.*, 2020, pp. 1–5.
[13] H. Wu, J. Chen, T. N. Nguyen, and H. Tang, "Lyapunov-guided delay-aware energy efficient offloading in IIoT-MEC systems," *IEEE Trans. Ind. Informat.*, vol. 19, no. 2, pp. 2117–2128, Feb. 2023.
[14] Q. Tang, R. Xie, F. R. Yu, T. Huang, and Y. Liu, "Decentralized computation offloading in IoT fog computing system with energy harvesting: A Dec-POMDP approach," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4898–4911, Jun. 2020.
[15] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Trans. Mobile Comput.*, vol. 18, no. 4, pp. 771–786, Apr. 2019.
[16] 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.
[17] H. Wu, K. Wolter, P. Jiao, Y. Deng, Y. Zhao, and M. Xu, "EEDTO: An energy-efficient dynamic task offloading algorithm for blockchain-enabled IoT-edge-cloud orchestrated computing," *IEEE Internet Things J.*, vol. 8, no. 4, pp. 2163–2176, Feb. 2021.
[18] X. Chen, C. Wu, Z. Liu, N. Zhang, and Y. Ji, "Computation offloading in beyond 5G networks: A distributed learning framework and applications," *IEEE Wireless Commun.*, vol. 28, no. 2, pp. 56–62, Apr. 2021.
[19] H. A. Shah, L. Zhao, and I.-M. Kim, "Joint network control and resource allocation for space-terrestrial integrated network through hierarchal deep actor–critic reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 70, no. 5, pp. 4943–4954, May 2021.
[20] N. Tian, H. Fang, J. Chen, and Y. Wang, "Nonlinear double-capacitor model for rechargeable batteries: Modeling, identification, and validation," *IEEE Trans. Control Syst. Technol.*, vol. 29, no. 1, pp. 370–384, Jan. 2021.
[21] 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.
[22] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*.
[23] Q. Li, S. Wang, A. Zhou, X. Ma, F. Yang, and A. X. Liu, "QoS driven task offloading with statistical guarantee in mobile edge computing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 1, pp. 278–290, Jan. 2022.
[24] T. de Bruin, J. Kober, K. Tuyls, and R. Babuška, "Experience selection in deep reinforcement learning for control," *J. Mach. Learn. Res.*, vol. 19, no. 9, pp. 1–56, 2018.
[25] J. M. Hancock, "Jaccard distance (Jaccard index, Jaccard similarity coefficient)," in *Dictionary of Bioinformatics and Computational Biology*. Hoboken, NJ, USA: Wiley, 2004.
[26] J. Li, Q. Liu, P. Wu, F. Shu, and S. Jin, "Task offloading for UAV-based mobile edge computing via deep reinforcement learning," in *Proc. IEEE/CIC Int. Conf. Commun. China (ICCC)*, 2018, pp. 798–802.
[27] Y. Wang and Z. Zhang, "Experience selection in multi-agent deep reinforcement learning," in *Proc. IEEE 31st Int. Conf. Tools Artif. Intell. (ICTAI)*, 2019, pp. 864–870.
[28] M. Li, J. Gao, L. Zhao, and X. Shen, "Deep reinforcement learning for collaborative edge computing in vehicular networks," *IEEE Trans. Cogn. Commun. Netw.*, vol. 6, no. 4, pp. 1122–1135, Dec. 2020.
[29] H. Xie and Z. Qin, "A lite distributed semantic communication system for Internet of Things," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 142–153, Jan. 2021.
[30] J. Cai, H. Fu, and Y. Liu, "Multitask multiobjective deep reinforcement learning-based computation offloading method for industrial Internet of Things," *IEEE Internet Things J.*, vol. 10, no. 2, pp. 1848–1859, Jan. 2023.
[31] M. Mukherjee, V. Kumar, Q. Zhang, C. X. Mavromoustakis, and R. Matam, "Optimal pricing for offloaded hard- and soft-deadline tasks in edge computing," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 7, pp. 9829–9839, Jul. 2022.
[32] B. Hazarika, K. Singh, S. Biswas, and C.-P. Li, "DRL-based resource allocation for computation offloading in IoV networks," *IEEE Trans. Ind. Informat.*, vol. 18, no. 11, pp. 8027–8038, Nov. 2022.
[33] B. Yamansavascilar, A. C. Baktir, C. Sonmez, A. Ozgovde, and C. Ersoy, "DeepEdge: A deep reinforcement learning based task orchestrator for edge computing," *IEEE Trans. Netw. Sci. Eng.*, vol. 10, no. 1, pp. 538–552, Jan./Feb. 2023.
[34] K. Li, X. Wang, Q. He, B. Yi, A. Morichetta, and M. Huang, "Cooperative multiagent deep reinforcement learning for computation offloading: A mobile network operator perspective," *IEEE Internet Things J.*, vol. 9, no. 23, pp. 24161–24173, Dec. 2022.
[35] Z. Yan, P. Cheng, Z. Chen, Y. Li, and B. Vucetic, "Gaussian process reinforcement learning for fast opportunistic spectrum access," *IEEE Trans. Signal Process.*, vol. 68, pp. 2613–2628, Apr. 2020.
[36] Q. He et al., "Routing optimization with deep reinforcement learning in knowledge defined networking," *IEEE Trans. Mobile Comput.*, early access, Jan. 9, 2023, doi: 10.1109/TMC.2023.3235446.
[37] D. Wu, T. Liu, Z. Li, T. Tang, and R. Wang, "Delay-aware edge-terminal collaboration in green Internet of Vehicles: A multiagent soft actor–critic approach," *IEEE Trans. Green Commun. Netw.*, vol. 7, no. 2, pp. 1090–1102, Jun. 2023.

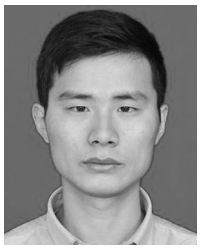
Kexin Li (Student Member, IEEE) received the B.S. degree in software engineering from Harbin University of Science and Technology, Harbin, China, in 2015, and the M.S. degree in computer technology from Northeastern University, Shenyang, China, in 2018, where she is currently pursuing the Ph.D. degree in computer application technology.

Her research interests include software-defined networking, edge computing, and machine learning.

Xingwei Wang received the B.S., M.S., and Ph.D. degrees in computer science from Northeastern University, Shenyang, China, in 1989, 1992, and 1998, respectively.

He is currently a Professor with the College of Computer Science and Engineering, Northeastern University. He has published more than 100 journal articles, books and book chapters, and refereed conference papers. His research interests include cloud computing and future Internet.

Prof. Wang has received several best paper awards.



Qiang He (Associate Member, IEEE) received the Ph.D. degree in computer application technology from Northeastern University, Shenyang, China, in 2020.

He is currently an Associate Professor with the College of Medicine and Biological Information Engineering, Northeastern University. He has authored or coauthored more than 50 journal articles and refereed conference papers. His research interests include social network analytic, machine learning, and data mining.

Qiang Ni (Senior Member, IEEE) received the B.Sc., M.Sc., and Ph.D. degrees in engineering from Huazhong University of Science and Technology, Wuhan, China.

He is a Professor of Communications and Networking with the School of Computing and Communications and Data Science Institute, Lancaster University, Lancaster, U.K., where he is also the Head of the Communications Research Group and the School's Director of International Partnership (previously was the School's Director of Postgraduate Studies and the School's Deputy Director of Research). His interests are wireless networks, communications, IoT, data analytics, and machine learning techniques, including energy-efficient green communications, cognitive radio networks, future wireless (5G/6G), intelligent communication techniques, SDN, edge/fog/cloud computing, big data analytics, AI and machine learning, IoT, cyber security, smart grids, sensor networks, vehicular networks, THz communication, quantum communication, quantum machine learning, and mobile positioning.

Prof. Ni is a Vice Chair of Big Data with Computational Intelligence SIG, IEEE ComSoc Technical Committee on Big Data. He is a Voting Member of the IEEE 1932.1 Standard. He is a Fellow of IET and Higher Education Academy.

Mingzhou Yang received the B.Sc. degree in computer science and technology from Shenyang University of Technology, Shenyang, China, in 2015, and the M.Sc. degree in computer software and theory and the Ph.D. degree in computer application technology from Northeastern University, Shenyang, in 2017 and 2022, respectively.

She is currently a Lecturer with Shenyang University of Technology. Her research interests include social network analysis, computational intelligence, and machine learning.

Schahram Dustdar (Fellow, IEEE) received the M.Sc. and Ph.D. degrees in business informatics from Johannes Kepler Universitat Linz, Linz, Austria.

He is a Full Professor of Computer Science (Informatics) with a focus on Internet Technologies heading the Distributed Systems Group, TU Wien, Wien, Austria. He has been the Chairman of the Informatics Section of the Academia Europaea since 9 December 2016. From 2004 to 2010, he was the Honorary Professor of Information Systems with the Department of Computing Science, University of Groningen (RuG), Groningen, The Netherlands. From December 2016 until January 2017, he was a Visiting Professor with the University of Sevilla, Seville, Spain, and from January until June 2017, he was a Visiting Professor with the University of California at Berkeley, Berkeley, CA, USA.

Dr. Dustdar is the recipient of the ACM Distinguished Scientist Award in 2009 and the IBM Faculty Award in 2012. He is an Associate Editor of IEEE TRANSACTIONS ON SERVICES COMPUTING, *ACM Transactions on the Web*, and *ACM Transactions on Internet Technology*, and on the editorial board of IEEE INTERNET COMPUTING. He is the Editor-in-Chief of *Computing* (an SCI-ranked journal of Springer). He has been a member of the IEEE Conference Activities Committee since 2016 and the Academia Europaea: The Academy of Europe, Informatics Section since 2013, and the Section Committee of Informatics of the Academia Europaea since 2015.