Digital Twin-Driven Intelligent Task Offloading for Collaborative Mobile Edge Computing

Yongchao Zhang, Jia Hu, Geyong Min

***Abstract*—Collaborative mobile edge computing (MEC) is a new paradigm that allows cooperative peer offloading among distributed MEC servers to balance their computing workloads. However, the highly dynamic workloads and wireless network conditions pose great challenges to achieving efficient task offloading in collaborative MEC. To address this challenge, digital twin (DT) has emerged as one promising solution by building a high-fidelity virtual mirror of the physical MEC to simulate its behaviors and help make optimal operational decisions. In this paper, we propose a DT-driven intelligent task offloading framework for collaborative MEC, where DT is employed to map the collaborative MEC system into a virtual space and optimize the task offloading decisions. We model the task offloading process as a Markov decision process (MDP) with the objective of maximizing the MEC system's total income from providing computing services, and then develop a deep reinforcement learning (DRL)-based intelligent task offloading scheme (INTO) to jointly optimize the peer offloading and resource allocation decisions. An efficient action refinement method is proposed to ensure that the action selected by the DRL agent is feasible. Experimental results show that our proposed approach can effectively adapt the task offloading decisions according to the dynamic environment, and significantly improve the MEC system's income through extensive comparison with three state-of-the-art algorithms.**

***Index Terms*—Edge computing; digital twin; task offloading; deep reinforcement learning**

I. Introduction

RECENT years have witnessed an exponential growth in the number of Internet-of-Things (IoT) devices being connected such as smart home devices and wearable devices. It is predicted that more than 30 billion IoT devices will be deployed in the world by 2027 [1]. Constrained by limited computing and energy resources, IoT devices usually cannot fulfill the stringent performance requirements of computation-intensive and delay-sensitive tasks. To meet the ever-increasing computational demands, mobile edge computing (MEC) has been proposed as a promising approach, which deploys cloud-like computing resources at the network edge, e.g., base stations (BSs), to offer real-time computing services for the nearby IoT devices [2], [3]. By providing computing capabilities in close proximity to the IoT devices, MEC enables timely responses to the delay-sensitive applications, and significantly reduces the traffic loads in the core networks. However, due to the spatial and temporal variations of dynamic computational demands, the workloads on different MEC servers may be severely imbalanced [4], [5]. The servers with heavy workloads suffer from serious performance degradation, while those with lots of unused computing resources have a low resource utilization. In order to improve the system performance, collaborative MEC has become a compelling solution by exploiting the cooperations among MEC servers [6], where peer offloading among MEC servers is enabled to balance the computing workloads. Such a collaborative mechanism can greatly enhance the utilization of computation resources and improve the system performance of MEC.

In collaborative MEC, it is crucial to design an effective task offloading scheme to satisfy the stringent performance requirements of computation tasks (e.g., service delay) constrained by the limited computation resources in MEC servers. However, it confronts two critical challenges. First, the heterogeneity of computation tasks (in terms of sizes, required resources, and tolerance delay), spatial diversity of computational demands, and limited computing resources of MEC servers have to be fully considered when optimizing the task offloading decisions. Second, the task offloading decisions are temporally coupled over consecutive time slots during the long-term system performance optimization process. However, the computational demands, wireless network environment, and resource availability of the MEC servers are all time-varying and highly stochastic, which are hard to be accurately predicted in practice.

There have been some works investigating the task offloading problem in collaborative MEC [6]–[9]. Li *et al.* [6] introduced social trust into the cooperative offloading to identify trustworthy edge servers, and proposed an online learning-based task offloading strategy. Sahni *et al.* [7] studied the multi-hop multi-task partial computation offloading problem to minimize the average completion time of all tasks. Despite these efforts, how to optimize the task offloading strategy online and guarantee its capability before practical application to proactively adapt to the system dynamics in practice is still a challenging problem [10]. To address this issue, digital twin (DT) has emerged as a desirable solution, which is a high-fidelity virtual representation of physical assets. Different from traditional simulation models, DT is a dynamic mapping that is able to adaptively update itself based on the real-time data from the physical world and historical data. In addition to monitoring the real-time system running status, DT can also simulate the behavior and evolvement trend of the physical entities, as well as accurately predict the effect

Manuscript received 01 December 2022; revised 19 May 2023; accepted 03 August 2023. This work was supported in part by UKRI Grant No. EP/X038866/1 and Horizon EU Grant No. 101086159. *(Corresponding authors: Jia Hu; Geyong Min.)*

Yongchao Zhang, Jia Hu and Geyong Min are with the Department of Computer Science, University of Exeter, Exeter, EX4 4QF, U.K. (e-mail: yz737@exeter.ac.uk; j.hu@exeter.ac.uk; g.min@exeter.ac.uk).

of specific control strategies on a real system, which supports the optimization of decision-making with a low cost.

With these attractive capabilities, DT has been applied in existing studies assisting the design of computation offloading and resource allocation scheme in MEC [11]–[14]. Huynh *et al.* [11] focused on the computation offloading problem in DT-aided wireless edge networks with the objective of minimizing task latency. Liu *et al.* [12] proposed a DT-assisted task offloading scheme based on deep reinforcement learning (DRL) where blockchain was applied to ensure data safety. Dong *et al.* [13] studied the joint user association, resource allocation, and offloading probabilities optimization problem in DT-empowered MEC system to minimize the energy consumption. However, the DT-assisted collaborative MEC has not yet been explored in the literature. To the best of our knowledge, this work is the first of its kind to incorporate DT in the optimization of task offloading for collaborative MEC.

In this paper, we focus on the intelligent task offloading problem in DT-driven collaborative MEC to maximize the MEC system's system income from providing computing services subject to the constraints of available computation resources and service delay. We leverage DT to create a high-fidelity digital replica of the physical collaborative MEC system, and deploy the proposed intelligent task offloading strategy in the DT domain for training, evaluation, and performance improvement. The major contributions of this paper are summarized as follows:

- We propose a new DT-driven collaborative MEC framework for achieving intelligent and efficient task offloading, where digital models of computation tasks, wireless channels, and edge serves are built to mirror the real MEC system. Given the proposed framework, we formulate a novel Markov decision process (MDP)-based income maximization problem while fully considering the temporal-spatial computational demands as well as heterogeneous and limited computing resources.
- We develop a novel DRL-based intelligent task offloading algorithm with the support of DT to jointly optimize the peer offloading and resource allocation decisions. An efficient action refinement approach is developed to map the action given by actor network to a valid one. The salient feature of our proposed algorithm is model-free, thus it does not require any prior system knowledge to make decisions.
- We conduct extensive experiments to validate the efficiency of the proposed DT-driven task offloading approach. The results show that our approach can effectively adapt to the changes in the MEC environment. Moreover, extensive comparison experiments with three state-of-the-art algorithms demonstrate that our proposed algorithm can significantly increase the MEC system's total income.

The rest of this paper is organized as follows. Related work is presented in Section II. Section III introduces the system model and formulates an income maximization problem. Section IV proposes an online DRL-based algorithm to solve the above problem. Section V conducts the performance evaluation and Section VI concludes this paper.

II. Related Work

In this section, we briefly summarize the existing works on the task offloading and DT in MEC.

A. Task Offloading in MEC

Task offloading and resource allocation in MEC is a popular topic and has been explored extensively. To alleviate the heavy computation workloads in the BSs located in hot spots, there are two main strategies, including computation offloading from BSs to remote clouds and collaborative computing among BSs. There have been some works considering the computation offloading between BSs and clouds. In [15], Ren *et al.* focused on the communication and computing resources allocation problem to minimize latency, and proposed an optimal task splitting strategy to determine the communication resource allocation, as well as a closed-form computing resource allocation strategy. Ning *et al.* [16] investigated the multi-user offloading and resource allocation problem, and proposed a heuristic algorithm to solve it. Yao *et al.* [17] proposed a blockchain-empowered task offloading for MEC to minimize the delay and energy consumption, and developed a truthful incentive mechanism to encourage each participant to contribute its computation resources. However, different from the remote clouds, in the collaborative MEC, the computing capacities of BSs are typically limited. In addition, the task offloading and resource allocation decisions are more coupled. These make the above works not applicable for collaborative MEC systems.

Some recent studies on collaborative MEC systems have also been conducted. Jia *et al.* [18] focused on how to balance workload between multiple cloudlets, and devised a scalable algorithm to minimize response time. In [19], Xiao *et al.* studied the task cooperative offloading among multiple computing nodes, and proposed a task forwarding algorithm based on the alternating direction method of multipliers. Wu *et al.* [20] focused on the load balancing and task admission problem and aimed at minimizing system cost. An online algorithm was designed based on the Lyapunov optimization technique. However, these works [18]–[20] did not consider the dynamics and heterogeneity of task demands in different MDs. Chen *et al.* [21] investigated the workload peer offloading among MEC-enabled small-cell BSs, and proposed an online peer offloading algorithm to minimize system delay cost. Although [21] considered that different MDs might have diverse task demands, it did not take the dynamics of task demand into account.

B. Digital Twin in MEC

Given the advantages in system simulation as well as verification and validation, DT has been successfully applied in the field of MEC. Huynh *et al.* [11] formulated the computation offloading in DT-assisted edge network as the latency minimization problem, and proposed an iterative optimization algorithm to solve this non-convex problem. Xu *et al.* [14] focused on the scenario of DT-empowered Internet of vehicle,

TABLE I
SUMMARY OF KEY NOTATIONS

Notation	Definition
$\mathcal{N}$	Set of N BSs
$\mathbf{U}_i$	Set of U_i IoT devices in the coverage of BS i
p_{ij}	Transmission power of IoT devices
$Q_{ij}^E(t)$	Queue length of the energy buffer in IoT devices
$e_{ij}(t)$	Estimated energy harvested by IoT devices
$z_{ij}(t)$	Data size of computation tasks
$v_{ij}(t)$	Processing density of computation tasks
$\delta_{ij}(t)$	Maximum tolerance delay of computation tasks
$g_{ij}(t)$	Award for completing computation tasks
$h_{ij}(t)$	Channel power gain between IoT devices and BSs
$\mathcal{G}_{ij}(t)$	Signal interference experienced by IoT devices
B	Wireless channel bandwidth
$\iota_{ij}(t)$	Wireless channel transmission rate of IoT devices
$d_{ij}^w(t)$	Wireless transmission delay of IoT devices
$C_i^{i'}(t)$	Data transmission rate between BS i and BS i'
$F_i(t)$	Computing capacity of BS i at time slot t
$x_{ij}(t)$	Computation offloading decision of each computation task
$y_{ij}^{i'}(t)$	Peer offloading decision of each computation task
$f_{ij}^{i'}(t)$	Computing resources that BSs allocate to each task

and applied deep Q-learning algorithm to learn the service offloading strategy. Sun *et al.* [22] employed DT to simulate the mobile devices and edge servers in MEC, and proposed a Lyapunov-based offloading scheme to minimize the offloading delay subject to the long-term migration cost. Dai *et al.* [23] built the digital network topology and stochastic task arrival models for industrial IoT systems, and formulated a stochastic optimization problem to maximize the energy efficiency. Although some efforts have been made to adopt DT in the task offloading and resource allocation for MEC, the application of DT in collaborative MEC is still largely unexplored. To fill this gap, we present a DT-driven collaborative MEC framework, and develop an intelligent algorithm to optimize the task offloading decisions.

III. System Model and Problem Formulation

A. DT-Driven Collaborative MEC

We consider a DT-driven collaborative MEC as shown in Fig. 1, which includes a physical network and a DT network. The physical network consists of N BSs indexed by $\mathcal{N} = \{1, 2, \cdots, N\}$, each of which is equipped with a MEC server to provide computing services. Some BSs are connected via high-speed wired optical fibers, which can be used to send task requests and results among BSs, enabling the cooperative peer offloading. In the coverage of each BS $i \in \mathcal{N}$, there are U_i IoT devices in total, the set of which is denoted as $\mathbf{U}_i$. Each IoT device is equipped with an energy harvesting module (e.g., solar and radio frequency harvester) to collect energy from the environment. The harvested energy will be stored in a local energy buffer (e.g., battery) to power data transmission and task processing. In this paper, we adopt a widely used time-slotted system (indexed by $t = 0, 1, \cdots$), which divides the operating period into several time slots with equal duration τ. The key notations are summarized in Table I.

At the beginning of time slot t, each MD generates a computation task to be offloaded. The DT network consists

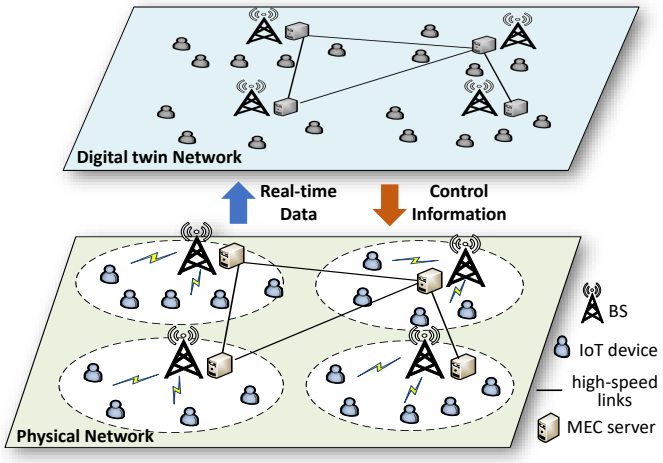


Fig. 1. DT-driven collaborative MEC

of the digital models of all the above physical entities (i.e., IoT devices and BSs), which monitors the real-time system running status as well as simulate and predict the behaviors of these physical entities. Specifically, a digital model is built for each IoT device to monitor their task offloading requests, energy harvesting rate, available energy resources, etc. The digital models of BSs are used to monitor the requests from IoT devices, network topology, and available computing resources in the MEC servers. In addition, the digital model of the wireless communication environment is created in the DT network to map the real-time status of wireless channels between IoT devices and BSs. The original constructions of these digital models are performed offline at the initialization time, while they will be updated online to capture the real-time changes of physical entities.

At each time slot t, for the IoT device j served by BS i, its digital model can be represented as a four-tuple, which is

$$\mathcal{D}_{ij}^K = \{p_{ij}, Q_{ij}^E(t), e_{ij}(t), \mathbf{O}_{ij}(t)\}, \tag{1}$$

where p_{ij} is the transmission power, $Q_{ij}^E(t)$ denotes the queue length of the energy buffer (i.e., the amount of stored energy) [24], $e_{ij}(t)$ is the estimated amount of energy that will be harvested during time slot t, and $\mathbf{O}_{ij}(t)$ is the computation task to be offloaded. For the task $\mathbf{O}_{ij}(t)$, it can be further represented by a tuple $< z_{ij}(t), v_{ij}(t), \delta_{ij}(t), g_{ij}(t) >$. $z_{ij}(t)$ is the data size (in bits), $v_{ij}(t)$ is the task processing density (i.e., the required CPU cycles to compute one bit data), $\delta_{ij}(t)$ denotes the maximum tolerance delay (in second) from generation to completion, and $g_{ij}(t)$ denotes the fee IoT devices would pay (i.e., task award) if the task is completed within the given deadline. The value of $v_{ij}(t)$ can be obtained by using the call-graph analysis method [25]–[27], which generates a call-graph of the computation task and analyzes in detail the computational complexity of each node and edge within this graph. Besides, we here consider the delay-sensitive computation tasks, whose maximum tolerance delay is shorter than the slot length τ [28], [29].

At the beginning of each time slot t, the offloading decision of task $\mathbf{O}_{ij}(t)$ is indicated by a binary variable $x_{ij}(t) \in \{0, 1\}$.

When $x_{ij}(t) = 1$, it will first be transmitted to its serving BS; otherwise, it will be discarded [29] and $x_{ij}(t) = 0$. Then, the maximum achievable communication rate of IoT device j served by BS i is as follows:

$$\iota_{ij}(t) = B\log_2(1 + \frac{p_{ij}h_{ij}(t)}{BN_0 + \mathcal{G}_{ij}(t)}), \tag{2}$$

where B is the wireless channel bandwidth, p_{ij} is the transmit power, $h_{ij}(t)$ denotes the channel power gain estimated by the DT of wireless communication network based on the current status and historical data, N_0 denotes Gaussian noise power spectrum density, and $\mathcal{G}_{ij}$ is the interference from other IoT devices. In this paper, we consider the intra-cell interference among IoT devices that are simultaneously transmitting to the same BS. Therefore, $\mathcal{G}_{ij}(t)$ can be calculated by

$$\mathcal{G}_{ij}(t) = \sum_{k \in \mathbf{U}_i \setminus j} x_{ik}(t)p_{ik}h_{ik}(t). \tag{3}$$

In addition, we assume that orthogonal frequency bands are allocated to adjacent BSs, so that the inter-cell interference can be eliminated [30], [31].

Then, if $\mathbf{O}_{ij}(t)$ is offloaded, its wireless communication delay is

$$d^w_{ij}(t) = \frac{z_{ij(t)}x_{ij}(t)}{\iota_{ij}(t)}. \tag{4}$$

When an IoT device offloads its computation task to BS, it should have sufficient energy to support the offloading operation. Thus, the following constraint should be satisfied:

$$p_{ij}(t)d^w_{ij}(t)x_{ij}(t) \le Q^E_{ij}(t). \tag{5}$$

Based on the above definitions, the queue length of the energy buffer in each IoT device evolves as follows:

$$Q^E_{ij}(t+1) = Q^E_{ij}(t) + e_{ij}(t) - x_{ij}(t)p_{ij}(t)d^w_{ij}(t). \tag{6}$$

At each time slot t, the digital model of BS i can be represented by a three-tuple $\{\mathbf{O}_{ij}(t)|_{j\in U_i}, C^{i'}_i(t)|_{i'\in\mathcal{N}\setminus\{i\}}, F_i(t)\}$. $C^{i'}_i(t)$ is the data transmission rate between BS i and BS i'. When $i = i'$, $C^{i'}_i(t)$ is equal to ∞ because the transmission delay within one BS is zero. Moreover, if there is no connection between BS i and BS i', $C^{i'}_i(t)$ equals 0. $F_i(t)$ is the available computing resources (i.e., CPU frequency) of the MEC server in BS i at time slot t.

Let the peer offloading decision from BS i to BS i' denote by a binary variable $y^{i'}_{ij}(t)$, which is

$$y^{i'}_{ij}(t) = \begin{cases} 1, \text{ BS } i \text{ offloads task } \mathbf{O}_{ij}(t) \text{ to BS } i'; \\ 0, \text{ otherwise.} \end{cases} \tag{7}$$

Note that if $x_{ij}(t) = 0$, $y^{i'}_{ij}(t)$ should also be 0. Then, if task $\mathbf{O}_{ij}(t)$ is determined to be transmitted to BS i', the transmission delay on the wired link is

$$d^c_{ij}(t) = \frac{z_{ij(t)}x_{ij}(t)y^{i'}_{ij}(t)}{C^{i'}_i(t)}. \tag{8}$$

And if task $\mathbf{O}_{ij}(t)$ is decided to be processed in its originally serving BS without peer offloading, $d^c_{ij}(t) = 0$ and $y^i_{ij}(t) = 1$.

Let $f^{i'}_{ij}(t)$ represent the CPU frequency that BS i' allocates to task $\mathbf{O}_{ij}(t)$. It is worth noting that if $y^{i'}_{ij}(t) = 0$, $f^{i'}_{ij}(t)$ is also equal to zero. Then, the computing delay of task $\mathbf{O}_{ij}(t)$ is

$$d^m_{ij}(t) = \frac{z_{ij}(t)v_{ij}(t)x_{ij}(t)}{\sum_{i'=1}^{N} f^{i'}_{ij}(t)}. \tag{9}$$

As the computing resources in each MEC server are limited, the resource allocation decisions should satisfy the following inequality:

$$\sum_{j=1}^{U_i}\sum_{i=1}^{N} f^{i'}_{ij}(t) \le F_{i'}(t), \forall i' \in \mathcal{N}. \tag{10}$$

B. Problem Formulation

To capture the dynamics in task demand and system environment, we here use a discrete-time MDP to model the task offloading problem. We first describe MDP briefly.

1) MDP: As a powerful tool for describing the stochastic environment, MDP has been widely used to conduct planning and decision-making while facing uncertainties. An MDP usually can be defined as a five-tuple, $(\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma)$, where $\mathbf{S}$ denotes the state space, $\mathbf{A}$ denotes the action space, $\mathbf{P}$ is the state transition function, $\mathbf{R}$ is the reward function and $\gamma \in [0, 1)$ is the discount factor. Define a policy π as a map from state to action, which is $\pi : \mathbf{S} \to \mathbf{A}$. At the beginning of the t-th slot, the system agent observes the current system $\mathbf{s}(t) \in \mathbf{S}$, and takes a corresponding action $\mathbf{a}(t) \in \mathbf{A}$ based on the predetermined policy π. After executing the selected action, the agent can receive an immediate reward $r(\mathbf{s}(t), \mathbf{a}(t)) \in \mathbf{R}$. Then, the system state transfers from $\mathbf{s}(t)$ to $\mathbf{s}(t+1)$ according to the state transition function $P(\mathbf{s}(t+1)|\mathbf{s}(t), \mathbf{a}(t)) \in \mathbf{P}$. For a given policy π, a state-value function $V^\pi(\mathbf{s}) = \mathbb{E}[\sum_{t=0}^{\infty}\gamma^t r(\mathbf{s}(t), \mathbf{a}(t))|\mathbf{s}(0) = \mathbf{s}, \pi]$ is defined to represent the expected cumulative discounted reward by adopting policy π when starting from state $\mathbf{s}$. The agent's aim is to find an optimal policy π^* to maximize the expected cumulative discounted reward, which is $\pi^* = \arg\max_\pi V^\pi(\mathbf{s}), \forall \mathbf{s} \in \mathbf{S}$.

2) MDP-based income maximization problem: Based on the MDP, an optimization problem is formulated to maximize the expected long-term income. The details of this optimization problem are presented below:

(1) State: At the beginning of each time slot t, the system state, $\mathbf{s}(t)$, consists of the following components:

- $z_{ij}(t)$: the data size of each task.
- $v_{ij}(t)$: the processing density of each task.
- $\delta_{ij}(t)$: the maximum tolerance delay of each task.
- $g_{ij}(t)$: the award of each task.
- $\iota_{ij}(t)$: the wireless channel transmission rate of each IoT device.
- $C^{i'}_i(t)$: the data transmission rate of each wired link.
- $Q^E_{ij}(t)$: the queue length of energy buffer in each IoT device.
- $F_i(t)$: the available computing capacity in each BS.

(2) Action: In the formulated MDP, the following three decisions need to be made: 1) whether the computation task

in each IoT device is offloaded to BSs for processing or not, 2) which offloaded tasks need to be further transmitted to other BSs via peer offloading; and 3) how many computing resources should be allocated to each offloaded task. Thus, the action, $\mathbf{a}(t)$, consists of the following components:

- $x_{ij}(t)$: the computation offloading decision of each task.
- $y_{ij}^{i'}(t)$: the peer offloading decision of each task.
- $f_{ij}^{i'}(t)$: the allocated computing resources to each task.

(3) State transition probability: At the given state $\mathbf{s}(t)$, if the selected action is $\mathbf{a}(t)$, the probability that the current state evolves into the new state $\mathbf{s}(t+1)$ is defined as $p(\mathbf{s}(t+1)|\mathbf{s}(t),\mathbf{a}(t))$. Notice that we target to design a model-free computation offloading strategy, which means that it does not require an explicit model to describe the state transition probability.

(4) Income function: After the action $\mathbf{a}(t)$ is executed, an immediate income, denoted by $r(t)$, will be received, which is the total award of all the tasks which are completed within the deadline. Specifically,

$$r(t)=\sum_{i=1}^{N}\sum_{j=1}^{U_i} g_{ij}(t)x_{ij}(t)\cdot\mathbf{1}\{d_{ij}(t)\le\delta_{ij}(t)\}, \tag{11}$$

where $d_{ij}(t)=d_{ij}^{w}(t)+d_{ij}^{c}(t)+d_{ij}^{m}(t)$ represents the completion duration of task $\mathbf{O}_{ij}(t)$, and $\mathbf{1}\{d_{ij}(t)\le\delta_{ij}(t)\}\in\{0,1\}$ indicates whether the task can be completed before its deadline or not. Note that since the output of computation tasks is often much smaller than their input size in practice, the transmission delay of result returning is negligible, which is a common setting in the related literature [26], [32].

Based on the above definitions, the income maximization problem (**IMP**) can be formulated as below:

$$\begin{aligned}
&\max \sum_{t=0}^{\infty}\gamma^t\sum_{i=1}^{N}\sum_{j=1}^{U_i} g_{ij}(t)x_{ij}(t)\cdot\mathbf{1}\{d_{ij}(t)\le\delta_{ij}(t)\},\\
&s.t.\ \ (5),(6),\text{and }(10),\\
&\quad C1: x_{ij}(t)\in\{0,1\},\ \forall j\in\mathbf{U}_i,\forall i\in\mathcal{N},\\
&\quad C2: y_{ij}^{i'}(t)\in\{0,1\},\ \forall j\in\mathbf{U}_i,\forall i,i'\in\mathcal{N},\\
&\quad C3: \sum_{i'=1}^{N} y_{ij}^{i'}(t)=x_{ij}(t),\ \forall j\in\mathbf{U}_i,\forall i\in\mathcal{N},\\
&\quad C4: 0\le f_{ij}^{i'}(t)\le F_{i'}(t),\ \forall j\in\mathbf{U}_i,\forall i,i'\in\mathcal{N}.
\end{aligned} \tag{12}$$

Here, C1 indicates that a task can be either offloaded to MEC or discarded. C2 states that a BS can transmit the extra tasks to other nearby BSs for computing or not. C3 ensures that each offloaded task can and only can be processed by one BS. C4 denotes the maximum computing resources each task can be allocated.

Remark: IMP is a stochastic optimization problem because the task demand, wireless channel state, and available computing capacity are all random variables. As **IMP** is a long-term optimization problem, the future information of the random variables is required to derive the optimal solutions to **IMP**. However, this information is usually unknown and hard to predict accurately. Therefore, it is highly challenging to optimize the peer offloading and resource allocation decisions without the requirement of future system information.

IV. DT-assisted Intelligent Task Offloading Algorithm

In this section, we design a DT-assisted intelligent task offloading algorithm to solve **IMP**, which applies DRL to optimize the peer offloading and resource allocation decisions through the interaction with DT network. Compared to the traditional approaches such as heuristic algorithms, DRL has two major advantages: (1) Thanks to the model-free feature, it can iteratively learn an optimal control policy through trial and error, without explicitly building a model of the system environment; (2) By leveraging the deep learning with the powerful representation ability to approximate the value/policy functions, DRL can effectively tackle the complicated optimization problem under a highly dynamic time-varying system. Next, we introduce the background of DRL and present the proposed DRL-based intelligent task offloading algorithm.

A. DRL Background

DRL is a powerful machine learning method that combines deep neural networks (DNNs) and reinforcement learning (RL) to handle complicated control tasks, such as robotics. For a conventional RL problem, an agent constantly interacts with the environment to learn an optimal control policy through trial and error, with the objective of maximizing the cumulative reward it received from the environment. In RL, Q-learning is a common model-free approach, which computes the Q-values of all the state-action pairs and stores them in a tabular form. The Q-value represents the expected cumulative reward obtained by taking a specific action a at a given state s, which is expressed by

$$\begin{aligned}
Q(s,a)=&\mathbb{E}[\sum_{t=0}^{\infty}\gamma^t r(t)|s_0=s,a_0=a],\\
&=r(s,a)+\gamma\sum_{s'}(P(s'|s,a)\max_{a'}Q(s',a')),
\end{aligned} \tag{13}$$

where s' is the next state after s.

Then, the temporal-difference method based on Bellman equation can be applied to update the Q-value, which is

$$Q(s,a)\leftarrow Q(s,a)+\varpi[r(s,a)+\gamma\max_{a'}Q(s',a')-Q(s,a)], \tag{14}$$

where ϖ represents the learning rate. However, the cost of computing and searching a Q-table is extremely high under a large state and action space. To solve this problem, DNNs can be integrated with RL to approximate the value and/or policy function, which is known as DRL. With the powerful representation learning of DNN, DRL can effectively learn a control strategy directly from the high-dimensional and complicated system, which presents great improvements over RL in terms of better policy and convergence time.

The DRL can be classified into two main categories, which are value-based DRL and policy-gradient-based DRL. However, for the value-based approach, such as deep Q-learning [33], a large action space would pose a big difficulty in selecting the optimal action from all the available actions. In such a context, we apply a policy-gradient-based DRL approach,

i.e., deep deterministic policy gradient (DDPG) algorithm, to solve **IMP**, which uses deep function approximators to learn the true Q-values and optimal policy. Unlike the value-based DRL approach, DDPG could directly output a deterministic action based on the current system state, and does not need to traverse all the states and actions. Thanks to this advantage, it has been widely and successfully applied in many fields, such as robotic control and process cooperative control [34], [35]. In the following, we devise an intelligent task offloading algorithm by taking advantage of DDPG algorithm.

B. Proposed Intelligent Task Offloading Algorithm

Define $Q(\mathbf{s}, \mathbf{a})$ as the Q-value for state-action pair $(\mathbf{s}, \mathbf{a})$, which can be regarded as the expected cumulative discounted income when action $\mathbf{a}$ is executed at state $\mathbf{s}$. Following the framework of DDPG, a critic network is introduced to approximate the Q-values. More specifically, a DNN is adopted to construct the critic network, which is represented by $Q(\mathbf{s}, \mathbf{a}) \approx Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^Q)$, where $\boldsymbol{\theta}^Q$ denotes the parameters of this critic network. In addition, another different DNN (called actor network) is created to generate the deterministic action, which is represented by $\hat{\mathbf{a}} = \phi(\mathbf{s}|\boldsymbol{\theta}^\phi)$, where $\boldsymbol{\theta}^\phi$ denotes the parameters of this actor network. At each slot, according to the current system state, the actor network could generate a deterministic action. However, this output action generally cannot be taken directly because it is continuous. In this case, we need to map it to a new feasible action in the discrete space. A common approach is Wolpertinger policy, whose main idea is finding κ-nearest-neighbor valid actions and then selecting the action with the maximum Q-value from these κ actions [36]. But, it is impractical to apply this strategy in this paper because looking for κ-nearest-neighbor valid actions is very complicated and time-consuming.

To address this problem, we propose a new fast action refinement approach to map the action generated by the actor network into a valid one. Specifically, we first map action $\hat{\mathbf{a}}(t)$ to a discrete action which satisfies constraints C1, C2 and C3 in **IMP**. Inspired by the idea of Wolpertinger policy, we select the nearest neighbor discrete action as follows:

$$\mathbf{a}'(t) = \arg\min_{\mathbf{a}(t)\in\mathcal{A}'(t)} |\mathbf{a}(t) - \hat{\mathbf{a}}(t)|, \tag{15}$$

where $\mathcal{A}'(t)$ denotes the set of valid actions satisfying constraints C1, C2 and C3. It can be seen that compared with the Wolpertinger policy which searches for κ-nearest-neighbor actions in $\mathcal{A}(t)$, our approach just needs to look for one action under the looser constraints. Hence, our approach can avoid the heavy computation cost in lookups.

Then, $\mathbf{a}'(t)$ is modified to satisfy constraint (5). Specifically, given offloading actions $\hat{x}_{ij}(t), \hat{y}_{ij}^{i'}(t) \in \mathbf{a}'(t)$ of task $\mathbf{O}_{ij}(t)$, if the IoT device has no sufficient energy to conduct computation offloading, $\hat{x}_{ij}(t)$ and $\hat{y}_{ij}^{i'}(t)$, $\forall i' \in \mathcal{N}$, are set to zero, which are

$$\hat{x}_{ij}(t) = \begin{cases} 0, & \text{if (5) is not satisfied,} \\ \hat{x}_{ij}(t), & \text{else,} \end{cases} \tag{16}$$

$$\hat{y}_{ij}^{i'}(t) = \begin{cases} 0, & \text{if (5) is not satisfied,} \\ \hat{y}_{ij}^{i'}(t), & \text{else.} \end{cases} \tag{17}$$

Algorithm 1: Fast action refinement approach

Input : Action $\hat{\mathbf{a}}(t)$ generated by actor network.
Output: A valid action $\mathbf{a}(t)$.

Obtain the nearest neighbor discrete action $\mathbf{a}'(t)$ according to (15);
Modify action $\mathbf{a}'(t)$ according to (16) and (17);
for *each BS* $i \in \mathcal{N}$ **do**
  Obtain the set of tasks, $\boldsymbol{\Upsilon}_i$, which are to be processed in BS i ;
  while *constraint (10) is not satisfied* **do**
    Search for the task $\mathbf{O}_{ij}(t)$ with the lowest value of $\varrho_{ij}(t)$ in $\boldsymbol{\Upsilon}_i$;
    Set $\hat{x}_{ij}(t) = 0$, $\hat{y}_{ij}^{i'}(t) = 0, \forall i' \in \mathcal{N}$;
    $\boldsymbol{\Upsilon}_i \leftarrow \boldsymbol{\Upsilon}_i \setminus \mathbf{O}_{ij}(t)$.

Finally, we refine action $\mathbf{a}'(t)$ to meet constraint (10). Recall that the optimization objective is to maximize the total system income. It is obvious that a higher income would be achieved when prioritizing the computation tasks with bigger awards but fewer resource demands. In such a context, we define the *processing priority* for each computation task as follows. Given the offloading decisions $x_{ij}(t)$ and $y_{ij}^{i'}(t)$, if the offloaded tasks are required to be completed before their deadlines, the minimum computing resource that the BS should allocate is

$$l_{ij}^{i'}(t) = \frac{z_{ij}(t) v_{ij}(t) x_{ij}(t) y_{ij}^{i'}(t)}{\delta_{ij}(t) - d_{ij}^{w}(t) - d_{ij}^{c}(t)}. \tag{18}$$

Then, the *processing priority* of each computation task is calculated as $\varrho_{ij}(t) = \frac{g_{ij}(t)}{l_{ij}^{i'}(t)}$. We can observe that if a task has either a higher award or fewer resource demands, its *processing priority* would become larger. Given $\varrho_{ij}(t)$ of all the offloaded tasks, if one BS does not have enough computing capacity (which is constrained by (10)), it would discard the tasks (i.e., let $\hat{x}_{ij}(t) = 0$, $\hat{y}_{ij}^{i'}(t) = 0, \forall i' \in \mathcal{N}$) with the lowest value of $\varrho_{ij}(t)$ until the computation demands of all the remaining tasks can be fulfilled. In this way, the proposed approach seeks to maximize the income as much as possible while satisfying the constraint (10). Algorithm 1 presents the details of the proposed action refinement approach.

Fig. 2 shows the training framework of the DT-assisted intelligent task offloading algorithm. Given the DT of real physical system, the intelligent task offloading algorithm is applied in the digital world to learn the optimal policy based on the feedback from DT. At each time slot t, the DRL agent takes the current system state given by the DT as the inputs of actor network, and determines the action to be taken according to Algorithm 1. Notably, before this step, in order to effectively explore the whole action space, a random noise would be added to the output action $\hat{\mathbf{a}}(t)$. Specifically, $\hat{\mathbf{a}}(t) = \phi(\mathbf{s}(t)|\boldsymbol{\theta}^\phi) + \eta(t)$, where $\eta(t)$ is a normally distributed random noise. After performing the selected action $\mathbf{a}(t)$ on the DT, the DT simulates the future evolvement of the system, and gives an immediate reward $r(t)$ and a new system state $\mathbf{s}(t+1)$. Then, based on the current state as well as the next

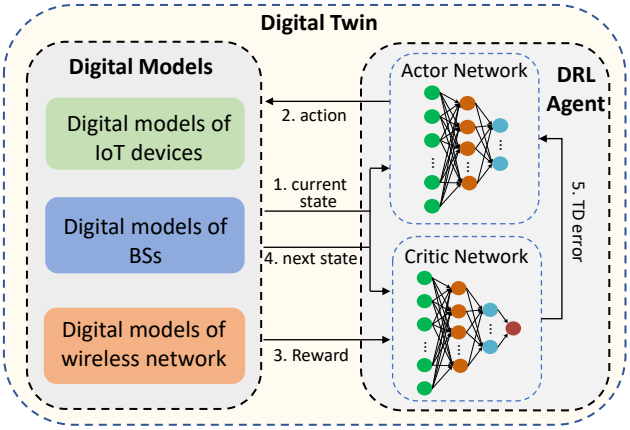


Fig. 2. Training framework of DT-assisted intelligent task offloading algorithm

state and reward given by DT, the DRL agent calculates the TD error to update the parameters of actor and critic networks.

In DDPG, critic and actor networks are trained in a supervised way. To improve the training stability, a replay memory $\mathbf{\Omega}$ is utilized to store the system experience. In each slot, agent stores the experience tuple $\boldsymbol{\vartheta}(t)$ in the replay memory $\mathbf{\Omega}$, where $\boldsymbol{\vartheta}(t) = (\mathbf{s}(t), \mathbf{a}(t), r(t), \mathbf{s}(t+1))$. If the replay memory does not have enough space to store the new experience records, the oldest experience record would be discarded. In the training process, a mini-batch samples, $\mathbf{\Lambda} \in \mathbf{\Omega}$, is chosen randomly from the replay memory for training the critic and actor networks. Besides, to further improve the learning stability, two separate target networks are introduced. Specifically, a copy of critic network, $Q'(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^{Q'})$, and a copy of actor network, $\phi'(\mathbf{s}|\boldsymbol{\theta}^{\phi'})$, are built respectively.

When training the critic network, $\boldsymbol{\theta}^{Q}$ is updated according to the Bellman equation as in Q-learning. The goal is to minimize the loss function $L(\boldsymbol{\theta}^{Q}) = (\rho - Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^{Q}))^2$, where $\rho = r(\mathbf{s}, \mathbf{a}) + \gamma Q'(\mathbf{s}', \phi(\mathbf{s}'|\boldsymbol{\theta}^{\phi'})|\boldsymbol{\theta}^{Q'})$, and $\mathbf{s}'$ is the next state after state $\mathbf{s}$. Based on the mini-batch experiences $\mathbf{\Lambda}$, $\boldsymbol{\theta}^{Q}$ can be updated by

$$\boldsymbol{\theta}^{Q} =: \boldsymbol{\theta}^{Q} + \alpha_Q \cdot \mathbb{E}_{(\mathbf{s},\mathbf{a},r(\mathbf{s},\mathbf{a}),\mathbf{s}')\in\mathbf{\Lambda}}[\nabla_{\boldsymbol{\theta}^{Q}} L(\boldsymbol{\theta}^{Q})], \tag{19}$$

where α_Q is the critic network's learning rate.

When training the actor network, DDPG uses a simple and low-computational deterministic policy gradient method where the main idea is to move the policy in the direction of $Q(\mathbf{s}, \mathbf{a})$'s gradient for maximizing the Q-value. Thus, by taking the derivative of $Q(\mathbf{s}, \phi(\mathbf{s}|\boldsymbol{\theta}^{\phi})|\boldsymbol{\theta}^{Q})$ with respect to $\boldsymbol{\theta}^{\phi}$, it can be obtained that $\nabla_{\boldsymbol{\theta}^{\phi}} Q(\mathbf{s}, \phi(\mathbf{s}|\boldsymbol{\theta}^{\phi})|\boldsymbol{\theta}^{Q}) = \nabla_{\boldsymbol{\theta}^{\phi}} \phi(\mathbf{s}) \cdot \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^{Q})|_{\mathbf{a}=\phi(\mathbf{s}|\boldsymbol{\theta}^{\phi})}$. Then, based on the mini-batch experiences $\mathbf{\Lambda}$, $\boldsymbol{\theta}^{\phi}$ can be updated by the following equation:

$$\boldsymbol{\theta}^{\phi} =: \boldsymbol{\theta}^{\phi} + \alpha_\phi \cdot \mathbb{E}_{\mathbf{s}\in\mathbf{\Lambda}}[\nabla_{\boldsymbol{\theta}^{\phi}} \phi(\mathbf{s}) \cdot \nabla_{\mathbf{a}} Q(\mathbf{s}, \mathbf{a}|\boldsymbol{\theta}^{Q})|_{\mathbf{a}=\phi(\mathbf{s}|\boldsymbol{\theta}^{\phi})}], \tag{20}$$

where α_ϕ is the actor network's learning rate.

When the network training is finished, the parameters of the target networks are updated as follows:

$$\boldsymbol{\theta}^{Q'} = \varepsilon_1 \boldsymbol{\theta}^{Q} + (1-\varepsilon_1)\boldsymbol{\theta}^{Q'}, \tag{21}$$

Algorithm 2: DRL-based intelligent task offloading algorithm (INTO)

Randomly initialize parameters $\boldsymbol{\theta}^{Q}$ and $\boldsymbol{\theta}^{\phi}$ in critic and actor networks;
Set parameters in target networks, i.e., $\boldsymbol{\theta}^{Q'} := \boldsymbol{\theta}^{Q}$ and $\boldsymbol{\theta}^{\phi'} := \boldsymbol{\theta}^{\phi}$;
Initialize replay memory $\mathbf{\Omega}$;
for $t = 0, 1, 2, \cdots$ **do**
  Generate a proto-action $\hat{\mathbf{a}}(t) = \phi(\mathbf{s}(t)|\boldsymbol{\theta}^{\phi}) + \eta(t)$ according to current system state;
  Obtain a valid action $\mathbf{a}(t)$ by using Algorithm 1;
  Perform action $\mathbf{a}(t)$, observe the immediate income $r(t)$ and new state $\mathbf{s}(t+1)$;
  Store experience tuple $\boldsymbol{\vartheta}(t) = (\mathbf{s}(t), \mathbf{a}(t), r(t), \mathbf{s}(t+1))$ in replay memory $\mathbf{\Omega}$;
  Randomly sample a mini-batch of experience tuples, $\mathbf{\Lambda}$, from $\mathbf{\Omega}$;
  Compute target value for critic network: $\rho = r(\mathbf{s}, \mathbf{a}) + \gamma Q'(\mathbf{s}', \phi(\mathbf{s}'|\boldsymbol{\theta}^{\phi'})|\boldsymbol{\theta}^{Q'})$;
  Update critic network parameters with (19);
  Update actor network parameters with (20);
  Update the parameters in target networks with (21) and (22).

$$\boldsymbol{\theta}^{\phi'} = \varepsilon_2 \boldsymbol{\theta}^{\phi} + (1-\varepsilon_2)\boldsymbol{\theta}^{\phi'}, \tag{22}$$

where $\varepsilon_1, \varepsilon_2 \ll 1$ are the parameters controlling the update rate. The proposed DRL-based intelligent task offloading algorithm, INTO, is given by Algorithm 2 in detail. By continuously interacting with the external environment, the proposed algorithm updates the actor and critic networks based on the received rewards via the stochastic gradient ascent method in (19) and (20), aiming to iteratively improve its learned policy. Therefore, instead of explicitly building a model of the underlying environment dynamics, the proposed algorithm is able to directly learn the optimal offloading policy through trial and error, which is considered model-free.

After completing the training phase, the DRL agent utilizes the well-trained actor network to make offloading decisions. At the beginning of each decision-making time step, the DRL agent observes the current system state and feeds it as inputs to the actor network to generate a possible action. Then, this generated action will be adjusted by using the proposed fast action refinement approach (i.e., Algorithm 1), to obtain a feasible action. Subsequently, the DRL agent performs the obtained feasible action and observes the resulting new state.

V. Performance Evaluation

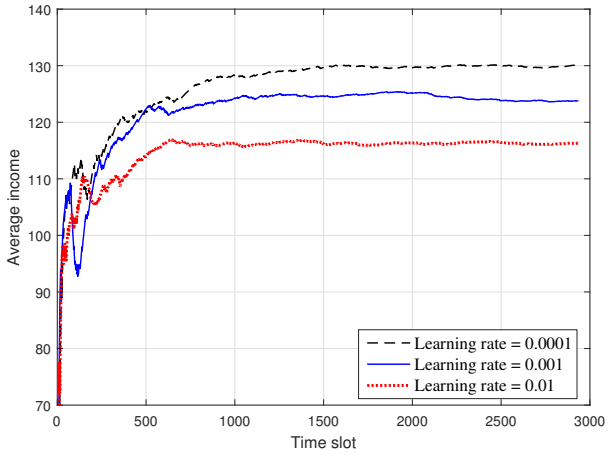
In this section, a series of experiments are conducted to evaluate the performance of INTO. We first investigate the impacts of several parameters on INTO, and then compare INTO with three baseline algorithms to show the effectiveness of INTO.

We consider a collaborative MEC system consisting of three BSs, which are BS-1, BS-2, and BS-3. Similar to

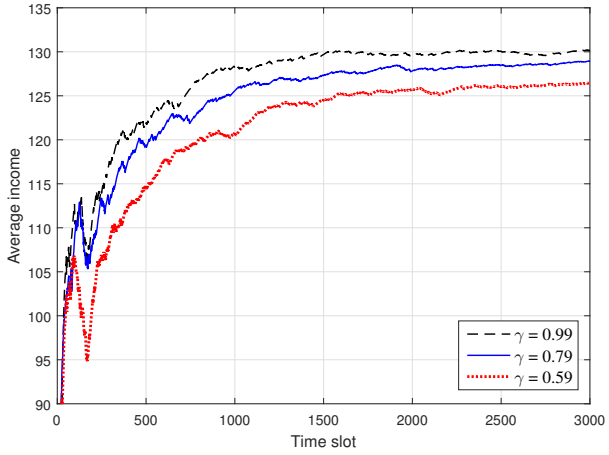
TABLE II
EXPERIMENTS PARAMETERS

Parameter	Value
The number of neurons in each hidden layer of critic network	(1500, 600)
The number of neurons in each hidden layer of actor network	(3000, 1000)
Learning rate of critic network	0.0001
Learning rate of actor network	0.0001
Update rate of critic network	0.001
Update rate of actor network	0.001
Batch size	64
Replay memory size	3000
Discount factor	0.99

[37], there are 30, 55, and 80 IoT devices in each BS respectively, and the data size of the generated task follows a uniform distribution, i.e., $z_{ij}(t) \sim U[50, 80]$ kbit. The processing density is uniformly distributed within $[3000, 8000]$ cycles/bit, the maximum delay $\delta_{ij}(t) \sim U[0.2, 2]$s [38], and the award of each task $g_{ij}(t) \sim U[1, 3]$. According to [39], the transmission power $p_{ij}(t)$ of each IoT device follows a uniform distribution on $[50, 180]$ mW. The harvested energy per slot $e_{ij}(t) \sim U[1, 1.5]$ mW. In addition, the channel bandwidth $B = 1$ MHz [40], the noise power spectrum density $N_0 = 1 \cdot 10^{-8}$ W/Hz, the computing capacity of each BS $F_i(t) = 5$ GHz, and the transmission rate between two different BSs $C_i^{i'}(t) = 100$ Mbps [41]–[43]. Both the critic and actor networks are constructed by an input layer, two hidden layers, and an output layer. Other parameters in this evaluation are listed in Table II.



(a) Impact of learning rate



(b) Impact of discount factor

Fig. 3. Average income with different learning rates and discount factors.

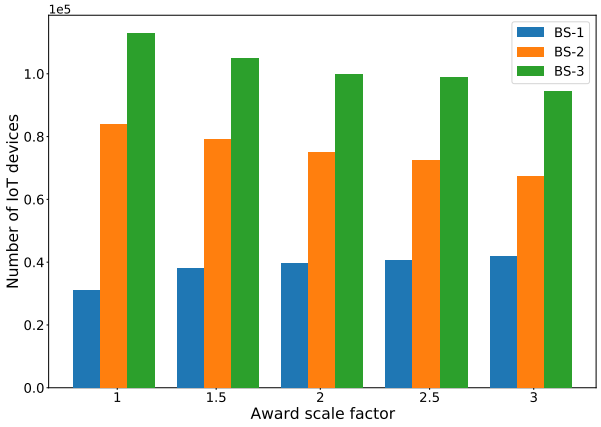
A. Parameter Analysis

We here present the impacts of the learning rate, discount factor, task award, task processing density, wireless channel gain, and harvested energy arrival rate on the performance of INTO.

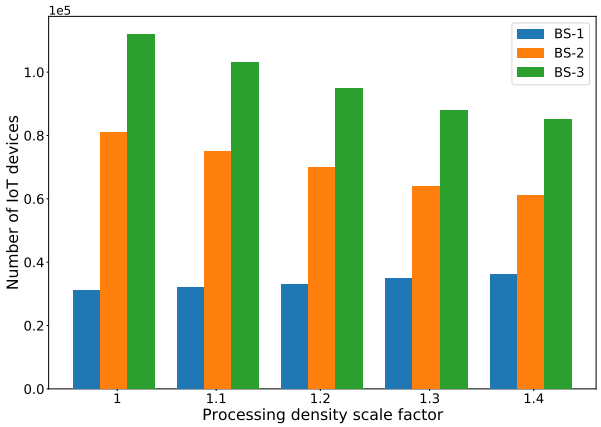
1) Impact of learning rate: Fig. 3(a) depicts the average income curve of INTO under different network learning rates. In this experiment, the learning rates of both critic and actor networks are set to 0.0001, 0.001, and 0.01, respectively. From Fig. 3(a), we can observe that a smaller learning rate has a higher income. This is because a smaller learning rate means a shorter learning step and it is more likely to achieve the global optimum. In contrast, a larger learning rate would lead to a longer learning step, and then the learning process may miss the global optimum. In addition, it can be seen that the income curve converges after about 600 slots when the learning rate is 0.01. And when the learning rates are 0.001 and 0.0001, the income curve converges after about 1000 slots and 1500 slots, respectively. It shows that decreasing the learning rate would slow down the convergence speed. The reason is that the reduction in the learning rate causes the learning step becomes shorter, naturally it requires more steps and time to converge.

2) Impact of discount factor: Fig. 3(b) plots the average income curve of INTO under different discount factors, where the discount factor γ is set to 0.59, 0.79, and 0.99, respectively. We can observe that as the discount factor rises, the average income also increases. This is because a smaller discount factor means that future income does not count much. As a result, the selected action just could improve the income in a very short period of time. While a larger discount factor endows INTO more foresight when updating strategy, then the income that future action brings is given more attention. Thus, the long-term income can be improved even more when adopting a larger discount factor.

3) Impact of task award: Fig. 4(a) shows the number of offloading IoT devices (i.e., the IoT devices whose tasks are decided to be offloaded) covered by each BS with different task awards. In this experiment, we set the task awards of the IoT devices covered by BS-1 as $\xi \cdot g_{1j}(t)$, where $\xi = 1, 1.5, 2, 2.5$ and 3, respectively. Meanwhile, the task awards of the rest IoT devices remain unchangeable. We can see from Fig. 4(a) that with the increase of award scale factor ξ, the number of offloading IoT devices in BS-1 rises, and those in both BS-2 and BS-3 reduce. This is because when the award scale factor increases, processing the tasks offloaded

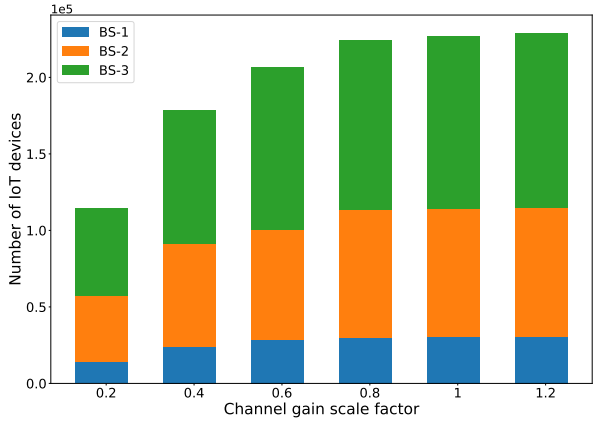


(a) Impact of task award

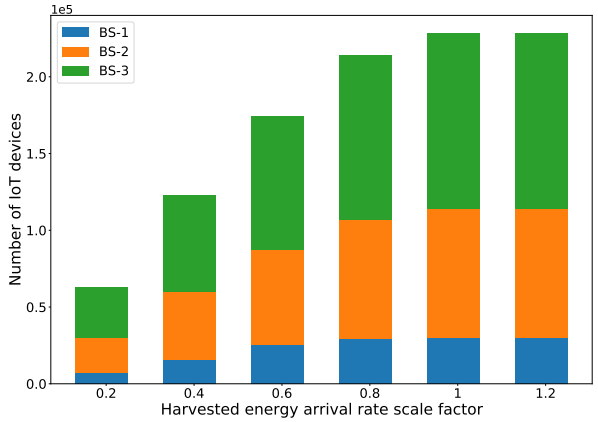


(b) Impact of task processing density

Fig. 4. Number of offloading IoT devices covered by each BS with different task awards and task processing densities.



(a) Impact of wireless channel gain



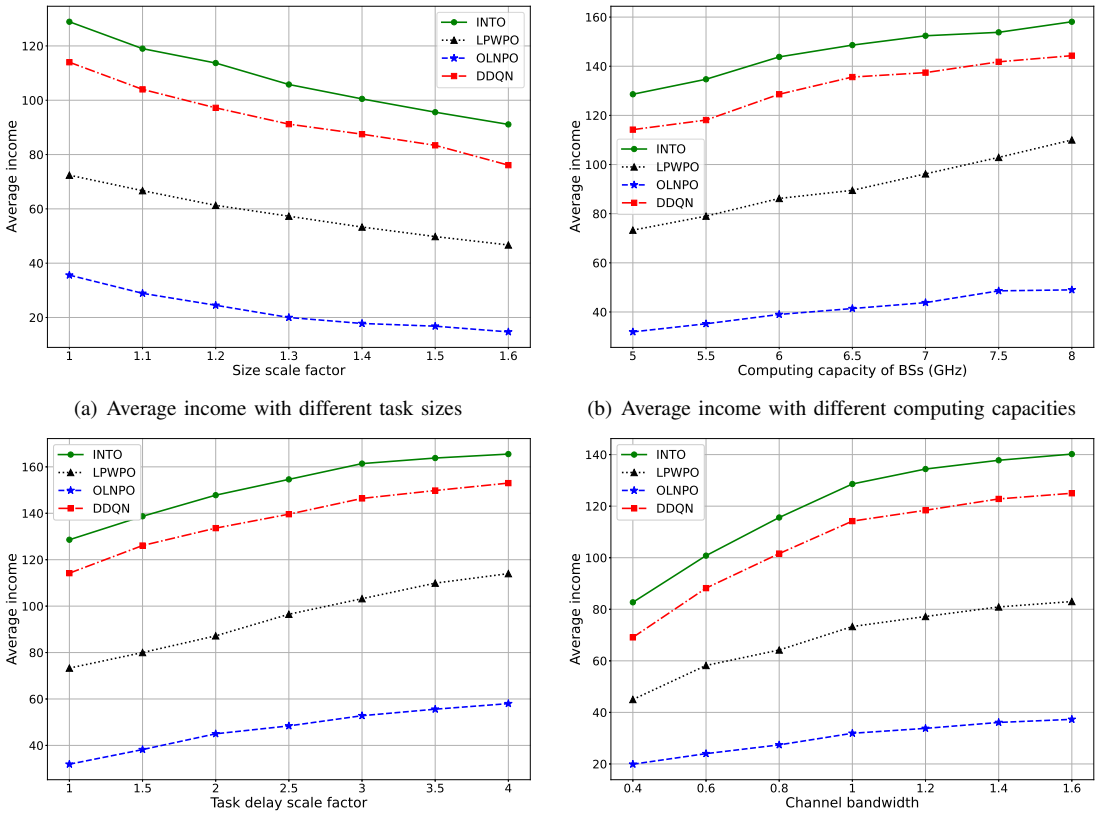
(b) Impact of harvested energy arrival rate

Fig. 5. Number of offloading IoT devices covered by each BS with different wireless channel gains and harvested energy arrival rates.

from the IoT devices in BS-1 would obtain more income. In this case, in order to maximize the income, the system would offload more tasks from the IoT devices in BS-1, and naturally fewer tasks in the IoT devices covered by BS-2 and BS-3 would be offloaded. It shows that INTO is able to effectively figure out and prioritize the tasks with larger awards to improve the income, and dynamically adjust the offloading decisions according to the changeable task award. In addition, in Fig. 4(a), we can also see that the number of offloading IoT devices in BS-1 is less than that in BS-2, and the number of offloading IoT devices in BS-3 is the largest. The reason is that the number of IoT devices in BS-1 is the smallest, and that in BS-3 is the largest.

4) Impact of task processing density: Fig. 4(b) shows the number of offloading IoT devices covered by each BS with different task processing densities. In this experiment, the task processing densities of the IoT devices covered by both BS-2 and BS-3 are set as $\zeta \cdot v_{2j}(t)$ and $\zeta \cdot v_{3j}(t)$, where $\zeta = 1, 1.1, 1.2, 1.3$ and 1.4, respectively. And the task processing densities of the IoT devices covered by BS-1 remain the same. We can see from Fig. 4(a) that as processing density scale factor ζ rises, the number of offloading IoT devices in BS-1 increases, and those in both BS-2 and BS-3 decrease. The reason is that with the rise of ζ, processing the same amount of data from the IoT devices covered by both BS-2 and BS-3 would require more computing resources. By contrast, the required computing resources of the IoT devices in BS-1 become fewer. With the aim of maximizing the income under the limited resources, more tasks in the IoT devices covered by BS-1 are intended to be offloaded, and the number of offloading IoT devices in both BS-2 and BS-3 would reduce accordingly. Fig. 4(b) indicates that INTO can dynamically adjust the offloading decisions to adapt to the changeable task processing density.

5) Impact of wireless channel gain: Fig. 5(a) shows the number of offloading IoT devices covered by each BS with different wireless channel gains. In this experiment, the wireless channel gain of each IoT device is set as $\sigma \cdot h_{ij}(t)$, where σ ranges from 0.2 to 1.2 with an increment of 0.2. From Fig. 5(a), we can see that with the increase of channel gain scale factor σ, the numbers of offloading IoT devices in the three BSs all increase. This is because as wireless channel gain rises, the transmission energy consumption in each IoT device would reduce, then more IoT devices would have enough energy to conduct computation offloading. In addition, we can also observe that the total number of offloading IoT devices almost remains unchangeable eventually even though channel gain is increasing. The reason is that the system is limited



(a) Average income with different task sizes

(b) Average income with different computing capacities

(c) Average income with different task delays

(d) Average income with different channel bandwidths

Fig. 6. Performance comparisons of INTO, OLNPO, LPWPO, and DDQN.

by its computing ability and it cannot support more tasks for computing. In this case, even if the channel gain keeps increasing, the total number of offloading IoT devices would stay the same.

6) Impact of harvested energy arrival rate: Fig. 5(b) shows the number of offloading IoT devices covered by each BS with different harvested energy arrival rates. In this experiment, the harvested energy arrival rate of each IoT device is set as $\rho \cdot e_{ij}(t)$, where ρ ranges from 0.2 to 1.2 with an increment of 0.2. From Fig. 5(b), it can be seen that as the harvested energy arrival rate scale factor ρ rises, the numbers of offloading IoT devices in the three BSs all increase. The reason is that with the increase of harvested energy arrival rate, the stored energy in each IoT device would become larger, and naturally more IoT devices can offload their tasks to MEC. Similarly, it can also be observed that with the increasing harvested energy arrival rate, the total number of offloading IoT devices almost does not change in the end. This is also because although IoT devices have more energy to conduct computation offloading, the computing resources in the system are still limited.

B. Comparison Experiments

Here, we compare INTO with three baseline algorithms in term of average income:

- Only local no peer offloading (OLNPO) [21]: Its main idea is that each BS can and only can process the tasks which are offloaded from the IoT devices within its coverage. In other words, each BS is not allowed to transmit its tasks to other BSs and help other BSs compute tasks.
- Local priority with peer offloading (LPWPO): In LPWPO, each BS prioritizes the tasks offloaded from the IoT devices within its coverage. Only when one BS can process all the tasks offloaded from the IoT devices it serves, can it help other BSs compute tasks
- Double deep Q-network (DDQN) [44], [45]: It is one kind of value-based deep reinforcement learning algorithm, which is used to solve (12). It contains two fully connected layers where the number of neurons is 1000 and 600, respectively. Both the learning rate and updating rate of the target network are set to 0.001.

Fig. 6(a) plots the average income of INTO, OLNPO, LPWPO, and DDQN with different task sizes. The task size of each IoT device is set as $\chi \cdot z_{ij}(t)$, where χ ranges from 1 to 1.6 with an increment of 0.1. It can be seen that when the size scale factor χ rises, the average income reduces. The reason is that increasing task size leads to a larger resource demand. In this case, the number of tasks that can be processed would reduce, and the average income would also decrease. Fig. 6(b) shows the average income of INTO, OLNPO, LPWPO, and DDQN with different computing capacities. From Fig. 6(b), we can observe that as computing capacity increases, the average income also rises. It is a reasonable result because a larger computing capacity means that more tasks can be

processed, naturally the average income would rise. Fig. 6(c) depicts the average income of INTO, OLNPO, LPWPO, and DDQN under varying task delays. The maximum delay for the computation task is set as $\delta_{ij}(t) \sim U[0.2 \cdot \psi, 2]s$, where ψ ranges from 1 to 4 with a step of 0.5. We can observe that as the task delay increases, the average income also rises. It is because when the maximum tolerance delay of computation tasks becomes larger, more tasks can be completed by each BS, leading to a higher system income. Fig. 6(d) presents the average income of INTO, OLNPO, LPWPO, and DDQN with different channel bandwidths. It is expected that with the increase of channel bandwidth, the average income will rise. Such a phenomenon can be seen in Fig. 6(d). It is due to that increasing channel bandwidth will reduce the transmission delay, resulting in a larger computing delay each task can tolerate. In this case, BSs are able to process more computation tasks before their deadlines, which yields a higher income.

In Fig. 6, we can clearly see that the average income of INTO is always larger than those of OLNPO, LPWPO, and DDQN. This is because INTO can make full use of the computing resources in the whole collaborative MEC system, and prioritize the tasks with higher awards and lower resource demands based on the whole system state. However, OLNPO cannot effectively utilize the spare resources in other BSs, thereby resulting in lots of waste of resources; LPWPO is unable to optimize the income from the view of the whole system because each BS prioritizes the tasks of the IoT devices it serves. For the DDQN, it will suffer from the curse of dimensionality problem when the action space is too large, which makes the learning inefficient.

VI. Conclusion

This paper investigates the task offloading problem in DT-driven collaborative MEC systems. To overcome the challenge brought by the unknown system dynamics, a novel DRL-based algorithm, INTO, is proposed. INTO does not require any prior system knowledge to make offloading decisions, which makes it more practical in the real world. In addition, INTO can optimize the long-term income in an online way, and automatically tune the offloading and resource allocation decisions as the task demand and system environment change. Extensive experiment results show that INTO can achieve a higher income and is able to be adaptive to the dynamic system environment.

References

[1] Ericsson, "Ericsson mobility report," 2022. [Online]. Available: https://www.ericsson.com/en/reports-and-papers/mobility-report/reports/june-2022

[2] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing¡ªa key technology towards 5g," *ETSI white paper*, vol. 11, no. 11, pp. 1–16, 2015.

[3] R. Schlegel, S. Kumar, E. Rosnes, and A. Graell i Amat, "Privacy-preserving coded mobile edge computing for low-latency distributed inference," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 3, pp. 788–799, 2022.

[4] S. Wang, X. Zhang, Z. Yan, and W. Wenbo, "Cooperative edge computing with sleep control under nonuniform traffic in mobile edge networks," *IEEE Internet of Things Journal*, vol. 6, no. 3, pp. 4295–4306, June 2019.

[5] X. He, S. Wang, and X. Wang, "Providing worst-case latency guarantees with collaborative edge servers," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2021.

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

[7] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multi-hop multi-task partial computation offloading in collaborative edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 32, no. 5, pp. 1133–1145, 2021.

[8] X. Lyu, C. Ren, W. Ni, H. Tian, and R. P. Liu, "Distributed optimization of collaborative regions in large-scale inhomogeneous fog computing," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 3, pp. 574–586, March 2018.

[9] X. He and S. Wang, "Peer offloading in mobile-edge computing with worst case response time guarantees," *IEEE Internet of Things Journal*, vol. 8, no. 4, pp. 2722–2735, 2021.

[10] L. Zhao, Z. Bi, A. Hawbani, K. Yu, Y. Zhang, and M. Guizani, "Elite: An intelligent digital twin-based hierarchical routing scheme for softwarized vehicular networks," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2022.

[11] D. Van Huynh, V.-D. Nguyen, S. R. Khosravirad, V. Sharma, O. A. Dobre, H. Shin, and T. Q. Duong, "Urllc edge networks with joint optimal user association, task offloading and resource allocation: A digital twin approach," *IEEE Transactions on Communications*, vol. 70, no. 11, pp. 7669–7682, 2022.

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

[13] R. Dong, C. She, W. Hardjawana, Y. Li, and B. Vucetic, "Deep learning for hybrid 5g services in mobile edge computing systems: Learn from a digital twin," *IEEE Transactions on Wireless Communications*, vol. 18, no. 10, pp. 4692–4707, 2019.

[14] X. Xu, B. Shen, S. Ding, G. Srivastava, M. Bilal, M. R. Khosravi, V. G. Menon, M. A. Jan, and M. Wang, "Service offloading with deep q-network for digital twinning-empowered internet of vehicles in edge computing," *IEEE Transactions on Industrial Informatics*, vol. 18, no. 2, pp. 1414–1423, 2022.

[15] J. Ren, G. Yu, Y. He, and G. Y. Li, "Collaborative cloud and edge computing for latency minimization," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 5, pp. 5031–5044, May 2019.

[16] 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 of Things Journal*, pp. 1–1, 2018.

[17] S. Yao, M. Wang, Q. Qu, Z. Zhang, Y.-F. Zhang, K. Xu, and M. Xu, "Blockchain-empowered collaborative task offloading for cloud-edge-device computing," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 12, pp. 3485–3500, 2022.

[18] M. Jia, W. Liang, Z. Xu, and M. Huang, "Cloudlet load balancing in wireless metropolitan area networks," in *IEEE INFOCOM 2016 - The 35th Annual IEEE International Conference on Computer Communications*, April 2016, pp. 1–9.

[19] Y. Xiao and M. Krunz, "Qoe and power efficiency tradeoff for fog computing networks with fog node cooperation," in *IEEE INFOCOM 2017 - IEEE Conference on Computer Communications*, May 2017, pp. 1–9.

[20] H. Wu, L. Chen, C. Shen, W. Wen, and J. Xu, "Online geographical load balancing for energy-harvesting mobile edge computing," in *2018 IEEE International Conference on Communications (ICC)*, May 2018, pp. 1–6.

[21] L. Chen, S. Zhou, and J. Xu, "Computation peer offloading for energy-constrained mobile edge computing in small-cell networks," *IEEE/ACM Transactions on Networking*, vol. 26, no. 4, pp. 1619–1632, Aug 2018.

[22] W. Sun, H. Zhang, R. Wang, and Y. Zhang, "Reducing offloading latency for digital twin edge networks in 6g," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 10, pp. 12 240–12 251, 2020.

[23] Y. Dai, K. Zhang, S. Maharjan, and Y. Zhang, "Deep reinforcement learning for stochastic computation offloading in digital twin networks," *IEEE Transactions on Industrial Informatics*, vol. 17, no. 7, pp. 4968–4977, 2021.

[24] M. Min, L. Xiao, Y. Chen, P. Cheng, D. Wu, and W. Zhuang, "Learning-based computation offloading for iot devices with energy harvesting," *IEEE Transactions on Vehicular Technology*, vol. 68, no. 2, pp. 1930–1941, Feb 2019.

[25] Z. He, K. Li, and K. Li, "Cost-efficient server configuration and placement for mobile edge computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 33, no. 9, pp. 2198–2212, 2022.

[26] J. Zheng, Y. Cai, Y. Wu, and X. Shen, "Dynamic computation offloading for mobile cloud computing: A stochastic game-theoretic approach," *IEEE Transactions on Mobile Computing*, vol. 18, no. 4, pp. 771–786, 2019.

[27] L. Yang, J. Cao, Y. Yuan, T. Li, A. Han, and A. Chan, "A framework for partitioning and execution of data stream applications in mobile cloud computing," vol. 40, no. 4, 2013.

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

[29] W. Chen, D. Wang, and K. Li, "Multi-user multi-task computation offloading in green mobile edge cloud computing," *IEEE Transactions on Services Computing*, pp. 1–1, 2018.

[30] A. Bozorgchenani, S. Maghsudi, D. Tarchi, and E. Hossain, "Computation offloading in heterogeneous vehicular edge networks: On-line and off-policy bandit solutions," *IEEE Transactions on Mobile Computing*, vol. 21, no. 12, pp. 4233–4248, 2022.

[31] L. Tan, Z. Kuang, J. Gao, and L. Zhao, "Energy-efficient collaborative multi-access edge computing via deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, pp. 1–10, 2022.

[32] Z. Wang, Y. Sun, D. Liu, J. Hu, X. Pang, Y. Hu, and K. Ren, "Location privacy-aware task offloading in mobile edge computing," *IEEE Transactions on Mobile Computing*, pp. 1–15, 2023.

[33] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[34] D. R. Song, C. Yang, C. McGreavy, and Z. Li, "Recurrent network-based deterministic policy gradient for solving bipedal walking challenge on rugged terrains," *arXiv preprint arXiv:1710.02896*, 2017.

[35] L. Gu, D. Zeng, W. Li, S. Guo, A. Zomaya, and H. Jin, "Deep reinforcement learning based vnf management in geo-distributed edge computing," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019, pp. 934–943.

[36] G. Dulac-Arnold, R. Evans, P. Sunehag, and B. Coppin, "Reinforcement learning in large discrete action spaces," *CoRR*, vol. abs/1512.07679, 2015. [Online]. Available: http://arxiv.org/abs/1512.07679

[37] C. Liu, M. Bennis, M. Debbah, and H. V. Poor, "Dynamic task offloading and resource allocation for ultra-reliable low-latency edge computing," *IEEE Transactions on Communications*, vol. 67, no. 6, pp. 4132–4150, June 2019.

[38] Y. Wu, K. Ni, C. Zhang, L. P. Qian, and D. H. K. Tsang, "Noma-assisted multi-access mobile edge computing: A joint optimization of computation offloading and time allocation," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 12, pp. 12 244–12 258, Dec 2018.

[39] S. Jošilo and G. Dán, "Wireless and computing resource allocation for selfish computation offloading in edge computing," in *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications*, April 2019, pp. 2467–2475.

[40] X. Zhang and J. Wang, "Heterogeneous statistical qos-driven power allocation for collaborative d2d caching over edge-computing networks," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, July 2019, pp. 944–953.

[41] Y. Chen, N. Zhang, Y. Zhang, X. Chen, W. Wu, and X. S. Shen, "Energy efficient dynamic offloading in mobile edge computing for internet of things," *IEEE Transactions on Cloud Computing*, pp. 1–1, 2019.

[42] H. Zhang, J. Guo, L. Yang, X. Li, and H. Ji, "Computation offloading considering fronthaul and backhaul in small-cell networks integrated with mec," in *2017 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, May 2017, pp. 115–120.

[43] Z. Xu, J. Tang, J. Meng, W. Zhang, Y. Wang, C. H. Liu, and D. Yang, "Experience-driven networking: A deep reinforcement learning based approach," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, April 2018, pp. 1871–1879.

[44] Q. Luo, C. Li, T. H. Luan, and W. Shi, "Collaborative data scheduling for vehicular edge computing via deep reinforcement learning," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 9637–9650, 2020.

[45] J. Gao, Z. Kuang, J. Gao, and L. Zhao, "Joint offloading scheduling and resource allocation in vehicular edge computing: A two layer solution," *IEEE Transactions on Vehicular Technology*, pp. 1–12, 2022.

Yongchao Zhang received the M.S. degree in computer technology from the School of Computer, Beijing Information Science and Technology University, China, in 2020. He is currently working toward the PhD degree in computer science at the University of Exeter. His research interests include wireless communication, smart grid, edge computing and deep reinforcement learning.

Jia Hu received the BEng and MEng degrees in electronic engineering from the Huazhong University of Science and Technology, China, in 2006 and 2004, respectively, and the PhD degree in computer science from the University of Bradford, UK, in 2010. He is a senior lecturer of computer science at the University of Exeter. His research interests include edge-cloud computing, resource optimization, applied machine learning, and network security.

Geyong Min received the BSc degree in computer science from the Huazhong University of Science and Technology, China, in 1995, and the PhD degree in computing science from the University of Glasgow, United Kingdom, in 2003. He is a professor of high performance computing and networking with the Department of Computer Science within the College of Engineering, Mathematics and Physical Sciences at the University of Exeter, United Kingdom. His research interests include computer networks, wireless communications, parallel and distributed computing, ubiquitous computing, multimedia systems, modeling and performance engineering.