MR-DRO: A Fast and Efficient Task Offloading Algorithm in Heterogeneous Edge/Cloud Computing Environments

Ziru Zhang, Nianfu Wang, Huaming Wu, *Member, IEEE*, Chaogang Tang, *Member, IEEE*, and Ruidong Li, *Senior Member, IEEE*

***Abstract*—With the rapid development of Internet of Things (IoT) and next-generation communication technologies, resource-constrained mobile devices (MDs) fail to meet the demand of resource-hungry and compute-intensive applications. To cope with this challenge, with the assistance of mobile-edge computing (MEC), offloading complex tasks from MDs to edge cloud servers (CSs) or central CSs can reduce the computational burden of devices and improve the efficiency of task processing. However, it is difficult to obtain optimal offloading decisions by conventional heuristic optimization methods, because the decision-making problem is usually NP-hard. In addition, there are shortcomings in using intelligent decision-making methods, e.g., lack of training samples and poor ability of migration under different MEC environments. To this end, we propose a novel offloading algorithm named meta reinforcement-deep reinforcement learning-based offloading, consisting of a meta-reinforcement learning (meta-RL) model, which improves the migration ability of the whole model, and a deep reinforcement learning (DRL) model, which combines multiple parallel deep neural networks (DNNs) to learn from historical task offloading scenarios. Simulation results demonstrate that our approach can effectively and efficiently generate near-optimal offloading decisions in IoT environments with edge and cloud collaboration, which further improves the computational performance and has strong portability when making offloading decisions.**

***Index Terms*—Deep neural network (DNN), Internet of Everything, mobile-edge computing (MEC), reinforcement learning, task offloading.**

I. Introduction

WITH the proliferation of various mobile devices (MDs), more and more resource-hungry applications, e.g., face recognition, autonomous driving, and augmented reality, have become an indispensable part of life. However, MDs, such as smartphones, tablet computers, and unmanned aerial vehicles (UAVs), usually have limited computing resources and constrained battery life, and thus the speed of processing compute-intensive tasks is insufficient to meet the delay and energy requirements of various Internet of Things (IoT) applications. In order to reduce service delay and save energy consumption, MDs often closely rely on the central cloud server (CS) to compute tasks in their daily operations. By offloading tasks from a local MD to the CS, the waiting time can be shortened and the battery life of the MD can also be extended. Despite the strong and scalable computing capacities of the cloud, it involves a large amount of data transmission when offloading computing tasks from MDs to the CS. In the case of insufficient bandwidth or network fluctuations, task offloading often brings high time costs. Meanwhile, with the increase of the number of MDs or tasks, the computing and communication delays also increase. Therefore, cloud computing cannot conform to the practical requirements for delay-sensitive tasks [1].

Benefiting from IoT and edge computing technologies, offloading compute-intensive tasks from MDs to the edge server (ES) at the edge of the network for execution has gradually matured. In this case, the remote cloud is no longer the only place for task offloading and application placement [2]. Edge computing can make full use of the hardware resources of the ES and alleviate the computing burden of the CS. Compared with the central CS, ES has relatively low computing and storage capabilities, and the integration degree of the heat dissipation and transfer equipment is lower than that of the CS. Resulting from that, the energy consumption of the ES is higher than that of the CS. Nonetheless, ESs are much closer to MDs, with low latency and more stable networks, which can greatly reduce the task offloading delay caused by the network and is suitable for latency-sensitive IoT applications.

In the practical application scenarios of mobile-edge computing (MEC) and mobile cloud computing (MCC), on the one hand, always offloading all tasks to the ES for execution is not advisable due to limited computing capacities of distributed ES; on the other hand, due to the high latency and insufficient bandwidth, offloading all tasks to the CS is not always beneficial [3], [4]. In addition, considering the heterogeneous resources of the MDs, ESs, and CSs, it is necessary for us to dynamically provide the optimal offloading decision for each

Manuscript received 21 September 2021; revised 19 October 2021; accepted 3 November 2021. Date of publication 8 November 2021; date of current version 6 February 2023. This work was supported by the National Natural Science Foundation of China under Grant 61801325 and Grant 62071327. *(Corresponding author: Huaming Wu.)*

Ziru Zhang is with the School of Mathematics, Tianjin University, Tianjin 300072, China (e-mail: zhangziru@tju.edu.cn).

Nianfu Wang is with the School of Mathematics, Harbin Institute of Technology, Harbin 15001, China (e-mail: 21s012049@stupid.hit.edu.cn).

Huaming Wu is with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: whming@tju.edu.cn).

Chaogang Tang is with the School of Computer Science and Technology, China University of Mining and Technology, Xuzhou 221116, China (e-mail: cgtang@cumt.edu.cn).

Ruidong Li is with the Institute of Science and Engineering, Kanazawa University, Kanazawa 920-1192, Japan (e-mail: liruidong@ieee.org).

Digital Object Identifier 10.1109/JIOT.2021.3126101

task according to different offloading scenarios. We intend to fully utilize all computing resources as well as obtain the maximum benefits. Moreover, the overhead time required for offloading decision making and the level of energy consumption also severely affect the real-world application deployment in edge computing environments. However, the total number of offloading decisions increases exponentially with the number of users and the number of tasks. Although this challenge can be solved well with the conventional optimization method for small-scale offloading scenarios, it will involve large amounts of calculation when the offloading scenario is complicated [5].

In recent few years, with the rapid development of artificial intelligence (AI), intelligent decision-making methods have become increasingly more popular [6]–[8]. Deep learning achieves high classifying accuracy when dealing with conventional classification problems. The offloading decision problem can be treated as a classification problem, in which the final decision can be regarded as a problem of classifying the tasks into three parts, namely, the local computing model, edge computing model, and cloud computing model, respectively. Through training the neural network, deep reinforcement learning (DRL) algorithms can quickly make offloading decisions in a specific edge/cloud computing environment [9]. However, in real-world IoT application scenarios, the number of users, the number of tasks, and the network conditions change frequently and dynamically. Thus, it is necessary to collect new training samples to retrain the neural network and make it suitable for the new offloading environment, which means that its migration ability is greatly limited. Instead, meta-reinforcement learning (meta-RL) can take advantage of the accumulated training experience to guide the new training process, so as to accelerate the completion of new training tasks [10]. Through the combination of DRL and meta-RL, we can both improve the portability of the model and reduce the total cost of the system.

Inspired by the above facts, this article designs a novel meta reinforcement-DRL-based offloading (MR-DRO) algorithm, where a meta-RL algorithm is adopted to give proper initial parameters for fast training and a DRL algorithm is applied to generate near-optimal offloading decisions. The main contributions of this article can be summarized as follows.

1) Considering the offloading performance in terms of response time and energy consumption during the offloading process, a system model is built in heterogeneous edge/cloud computing environments with multiple mobile terminal users, different volumes of data, and different scales of task workloads. To this end, we formalize the offloading decision-making issue as an optimization problem and attempt to solve it in an intelligent and effective way.
2) We design a novel offloading framework composed of a meta-RL model and a DRL model. For the former, we adopt the Reptile algorithm to train several neural networks, by which we can avoid the second-order gradient calculation process, thereby reducing the cost of decision making. Using the initial parameters generated by the meta-RL model, we can greatly improve the initial accuracy of the decision-making model and increase the algorithm portability. For the latter, we use multiple parallel deep neural networks (DNNs) to determine when and where each task should be offloaded, which is achieved by the cycle of generating labeled samples and updating the parameters of DNNs.
3) We conduct comprehensive experiments in real-world MEC environments to evaluate the proposed MR-DRO approach, which achieves superior offloading performance when compared with other offloading-decision schemes. Moreover, it can make the DNNs reach a state of convergence and significantly improve the offloading accuracy, while being able to adapt fast to new scenarios.

The remainder of this article is organized as follows. In Section II, we discuss the related works. Section III first develops the system model and then formulates the offloading-decision problem. Section IV presents the details of the proposed algorithm. Performance evaluation of MR-DRO is discussed in Section V. In Section VI, we conclude this article and point out several potential directions.

II. Related Work

In recent years, a large number of offloading-decision schemes have been proposed to maximize the offloading performance in heterogeneous MEC and MCC environments, which are mainly based on conventional offloading-decision approaches and intelligent offloading-decision approaches as listed in Table I.

A. Conventional Offloading-Decision Approaches

There are several studies dealing with the offloading problem in the environment with poor network stability. eTime [11] was a Lyapunov optimization-based method, which can preload data when the network connection is poor, and give priority to offloading delay-sensitive tasks in the case of limited bandwidth. Thus, it can be applied to most applications while saving 25%–30% of energy consumption by simulating actual offloading scenes. Li *et al.* [12] adopted the Lyapunov optimization method to establish a queueing model to simulate the offloading process and minimize the queue length, so as to achieve a relatively low overall consumption of offloading decisions. Haber *et al.* [13] transformed the original decision-making issue into a nonconvex programming mathematical problem by establishing an appropriate task offloading mathematical model and then converted it into a series of convex problems through a continuous convex approximation programming method. To achieve energy-efficient task assignment when combing MEC offloading and Device-to-Device (D2D) offloading, Yu *et al.* [14] proposed TA-MCTS, a Monte Carlo tree search-based approach for solving the optimal offloading-decision problem.

The computing tasks for specific IoT applications in a heterogeneous MEC/MCC environment can also be viewed as a workflow, so that offloading decisions can be made by using graph theory, game theory, and genetic algorithm (GA).

TABLE I
QUALITATIVE COMPARISON OF THE CURRENT LITERATURE

Categories	Offloading Schemes	Theories	Mode	Architectural Properties		Decision Objectives		Fast Adaptability
				MCC	MEC	Latency	Energy	
Conventional Offloading Decisions	eTime [11]	Lyapunov Optimization	Full	✓	✗	✗	✓	✗
	OOD [12]	Lyapunov Optimization	Full	✓	✗	✓	✗	✗
	SCA-based Scheme [13]	Successive Convex Approximation	Full	✗	✓	✗	✓	✗
	TA-MCTS [14]	Monte Carlo Tree Search Optimization	Partial	✗	✓	✗	✓	✗
	MCOP [15]	Graph Theory	Partial	✓	✓	✓	✓	✗
	LARAC-based Scheme [16]	Graph Theory	Partial	✓	✗	✗	✓	✗
	K-LARAC & M-LARAC [17]	Lagrangian Relaxation-based Aggregate Cost	Partial	✓	✗	✓	✓	✗
	COM [18]	Genetic Algorithm	Partial	✓	✓	✓	✓	✗
	F-SGA & C-SGA [19]	Stalberg Game Theory	Partial	✓	✗	✓	✗	✗
	MDP-based Scheme [20]	Markov Decision Process	Partial	✓	✓	✓	✓	✗
	EMOP [21]	Markov Decision Process	Partial	✓	✗	✗	✓	✗
	Wu *et al.* [22]	Queueing Theory	Partial	✓	✗	✓	✓	✗
Intelligent Offloading Decisions	Li *et al.* [23]	Deep Learning	Full	✗	✓	✗	✗	✗
	Neurosurgeon [24]	Deep Neural Networks	-	✓	✓	✓	✓	✗
	QL-JTAR [25]	Q-Learning	Partial	✗	✓	✓	✓	✗
	DIOS [26]	Deep Imitation Learning	Partial	✗	✓	✓	✗	✗
	DDLO [27]	Distributed Deep Learning	Partial	✗	✓	✓	✓	✗
	DDTO [5]	Distributed Deep Learning	Partial	✓	✓	✓	✓	✗
	DMRO [28]	Deep Meta Reinforcement Learning	Partial	✗	✓	✓	✓	✓
	MRLCO [29]	Meta Reinforcement Learning	Partial	✗	✓	✓	✗	✓
	Our MR-DRO	Reinforcement Learning & Meta Learning	Partial	✓	✓	✓	✓	✓

Wu *et al.* [15] transformed the offloading environment into a weighted graph model and proposed the MCOP algorithm based on the graph theory. Using this algorithm, they successfully divided the task into the local part and the edge part. Zhang and Wen [16] subdivided the tasks in MDs and transformed the subdivided tasks into topological models in accordance with the logical relationship, and provided offloading decisions, respectively, for scenes without offloading restrictions and general offloading scenarios. Haghighi and Moayedian [17] took time delay and energy consumption factors into consideration and proposed the LARAC algorithm to find the shortest path in the graph-based model, by which they find the near-optimal solution of the task offloading decisions. Xu *et al.* [18] comprehensively considered the execution time and energy consumption for IoT devices in the scene combining MEC and MCC. They represented the overall offloading scheme through an ordered array and iterate the possible offloading solutions through nondominated sorting GA III (NSGA-III), thus obtaining the near-optimal solution. Li *et al.* [19] innovated on the basis of the Stalberg game model and designed F-SGA and C-SGA algorithms specifically for delay-sensitive and compute-intensive applications, respectively. When the model reaches the game equilibrium point, the approximate optimal solution of the offloading decision can be obtained.

The Markov decision process (MDP) is also a theoretical tool widely used for offloading decision making. In the MEC scenario, Alasmari *et al.* [20] proposed the offloading scheme by applying the MDP, which improved the decision-making level by more than 17.47%. Terefe *et al.* [21] proposed the EMOP algorithm on the basis of MDP, and used discrete-time Markov chains to represent the wireless channel of MDs. This algorithm can solve the offloading decision problem when there are multiple edge clouds that can be used for offloading. In the MCC scenario, Wu and Wolter [22] established a queueing model for the decision-making problem. They represented the model delay by 2-D Markov chain and generated the offloading decision by an M/G/1-FCFS queue model.

Relying on a variety of conventional optimization methods, we can generate proper offloading decisions, however, these methods usually involve a large number of matrix operations and gradient operations. It is known that the offloading-decision problem is NP-hard, thereby brute force algorithms are unsuitable for such problem, especially when the scale of the problem is large, the time delay and energy consumption caused by decision making will become unacceptable. Therefore, we need to design an efficient offloading-decision algorithm to replace conventional heuristic algorithms. It has recently become one of the main research directions to propose an algorithm that can give offloading decisions in an intelligent manner.

B. Intelligent Offloading-Decision Approaches

Due to the numerous advantages of deep learning, e.g., immediacy and portability, it has broad application prospects in the field of edge computing. Therefore, many studies have tried to integrate AI methods into MEC and MCC to make offloading decisions [30].

Li *et al.* [23] used the deep learning method to tackle the offloading-decision issue. They trained the DNN according to the historical offloading decision data before making decisions and also verified through examples that the offloading decisions given by deep learning are better than conventional optimization methods. Kang *et al.* [24] used eight different mobile intelligent applications to verify the reliability of the deep learning approach and proved that this scheme reduces 59.5% of energy consumption. In addition, Dab *et al.* [25] proposed the QL-JTAR algorithm based on *Q*-learning. This algorithm comprehensively considered the resource allocation problem and task offloading decision problem in edge computing and proved that the offloading decisions obtained through the algorithm have high accuracy. Yu *et al.* [26] trained the neural network through deep imitation learning and made decisions in MEC and MCC scenarios, thereby improving the training speed of the model and accuracy of the decision.

Due to the particularity of task offloading in heterogeneous computing environments, the samples used for training DNNs are always difficult to obtain, especially for large-scale offloading-decision problems. To tackle this challenge, Huang *et al.* [27] proposed a DDLO algorithm based on DRL, through multiple parallel DNNs to train the model and update the training data set. This algorithm can improve the precision of the data set and update the parameters of DNNs simultaneously, thereby reducing the dependence on training samples. Wu *et al.* [5] proposed a DDTO algorithm in a heterogeneous MEC and MCC environment, where ES and CS can collaborate in computing and proved that the error of the offloading decision made by this algorithm can be controlled within 10%.

Although DNN can quickly generate offloading decisions, when the number of users or tasks changes, the number of nodes in the input layer and output layer of DNNs often cannot be applied to the new environment so that DNNs are required to be retrained. To tackle the aforementioned challenges, Qu *et al.* [28] proposed DMRO, a task offloading algorithm based on deep meta-RL. When faced with a new offloading environment, DMRO can generate appropriate initial parameters of DNNs, so as to significantly accelerate the subsequent training speed and improve the portability of the model. Wang *et al.* [29] proposed the MRLCO algorithm on the basis of meta-RL, which reduces the amount of calculation caused by the second-order gradient in model-agnostic meta-learning (MAML) without significantly reducing the accuracy of offloading decisions.

Most of the aforementioned work attempted to reduce the system latency or energy consumption in MEC/MCC environments while neglecting the fast adaptability of task offloading models. In this article, we concentrate on enhancing the robustness and portability of the model, enabling edge computing technology to be better applied in real life. We design an efficient intelligent decision-making approach to generate a near-optimal offloading decision with a small amount of calculation. It serves as an approximation algorithm, improves the speed of decision making, as well as reduces the waiting time of MDs. Moreover, we can quickly and intelligently provide near-optimal offloading decisions when facing different MEC scenarios.

III. System Model and Problem Formulation

In this section, we first build the system model composed of the local computing model, edge computing model, and cloud computing model. Then, we formulate the task offloading decision-making problem as an optimization problem. For convenience, the major notations used in this article are summarized in Table II.

TABLE II
Important Notations Used in This Article

Notation	Description
w_{nm}	The amount of data for $task_{nm}$
b_n	The bandwidth of the n_{th} MD
$x_{nm}^{(1)}$	An indicator determines whether $task_{nm}$ is offloaded
$x_{nm}^{(2)}$	An indicator determines whether $task_{nm}$ is offloaded to ES or CS
σ	The number of instructions that CPU needs to calculate
e_t	The energy consumed to transmit a unit of data
f_l	The task processing rate of the MD
f_e	The task processing rate of the ES
f_c	The task processing rate of the CS
E_{nm}^{local}	The energy consumed when $task_{nm}$ is executed on the MD
E_{nm}^{edge}	The energy consumed when $task_{nm}$ is offloaded to the ES
E_{nm}^{cloud}	The energy consumed when $task_{nm}$ is offloaded to the CS
T_{nm}^{local}	The response time taken when $task_{nm}$ is executed locally
T_{nm}^{edge}	The response time taken when $task_{nm}$ is offloaded to the ES
T_{nm}^{cloud}	The response time taken when $task_{nm}$ is offloaded to the CS
ϵ_l	The energy consumption by the MD for per unit of workload
ϵ_e	The energy consumption by the ES for per unit of workload
ϵ_c	The energy consumption by the CS for per unit of workload

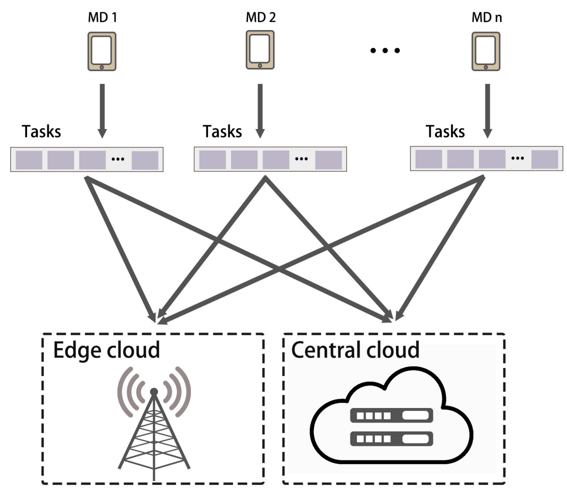


Fig. 1. System model of task offloading in a heterogeneous edge/cloud computing environment.

A. System Model

As depicted in Fig. 1, we consider a heterogeneous collaborative edge/cloud computing environment, which integrates MDs for local computing, ES for edge computing, and CS for cloud computing. Without loss of generality, we denote the set of MDs as $\mathcal{N} = \{1, 2, \ldots, N\}$, assuming that there are N mobile users and the set of tasks as $\mathcal{M} = \{1, 2, \ldots, M\}$, assuming that each MD has M tasks to be offloaded. Each user may have multiple tasks, and each task can choose to be executed locally or to be offloaded, either to the ES or the CS for computing. The data size of the task to be offloaded is often different, we assume that w_{nm} is the amount of data to be offloaded for task$_{nm}$, i.e., the mth task of the nth MD.

To clearly represent the offloading decision for each task, we set a pair of indicators, namely, $x_{nm}^{(1)} \in \{0, 1\}$ and $x_{nm}^{(2)} \in \{0, 1\}$. For any task$_{nm}$, $x_{nm}^{(1)}$ is the indicator that decides whether to offload or not, which is denoted as

$$x_{nm}^{(1)} = \begin{cases} 1, & \text{if task}_{nm} \text{ is processed locally on MD} \\ 0, & \text{if task}_{nm} \text{ is offloaded to the ES/CS} \end{cases} \tag{1}$$

where $x_{nm}^{(1)} = 1$ if task$_{nm}$ is not offloaded and only executed locally on the MD; otherwise, $x_{nm}^{(1)} = 0$, if task$_{nm}$ is offloaded to the server.

In the same way, $x_{nm}^{(2)}$ decides where to offload, that is, either to the ES or the CS, which is denoted as

$$x_{nm}^{(2)} = \begin{cases} 1, & \text{if task}_{nm} \text{ is offloaded to the ES} \\ 0, & \text{if task}_{nm} \text{ is offloaded to the CS} \end{cases} \tag{2}$$

where $x_{nm}^{(2)} = 1$ only if task$_{nm}$ will be offloaded to the ES; otherwise, $x_{nm}^{(2)} = 0$ only if it is offloaded to the CS.

Both response time and energy consumption are taken into consideration during the offloading process with the combination of MEC and MCC. When a task is selected either to run on an MD, offloaded to the ES or offloaded to the CS, the offloading performance in terms of response time and energy consumption corresponding to the aforementioned offloading decisions is different. We will elaborate on the local computing model, edge computing model, and cloud computing model, respectively.

1) Local Computing Model: Once task$_{nm}$ is chosen to be executed locally on the MD, we have $x_{nm}^{(1)} = 1$.

The response time required for calculating task$_{nm}$ locally on the MD can be described as

$$T_{nm}^{\text{local}} = \frac{\sigma w_{nm}}{f_l} \tag{3}$$

where f_l is denoted as the computational capacity (i.e., CPU cycles per second) of user n. It is assumed that the CPU needs to run σ instructions to handle per unit of the task. Because the number of instructions that a task needs to process will not change whether it is in the MD, the ES, or the CS, the coefficient σ also holds for the other two offloading cases.

It is easy to know that the energy required for computing task$_{nm}$ on the MD is calculated by

$$E_{nm}^{\text{local}} = \sigma \epsilon_l w_{nm} \tag{4}$$

where we assume that each MD needs the average energy ϵ_l to process an instruction.

Besides, the total response time and total energy consumption by the nth user to perform on the MD can be calculated as follows, respectively:

$$T_n^{\text{local}} = \sum_{m=1}^{M} \left[x_{nm}^{(1)} \cdot T_{nm}^{\text{local}} \right] \tag{5}$$

$$E_n^{\text{local}} = \sum_{m=1}^{M} \left[x_{nm}^{(1)} \cdot E_{nm}^{\text{local}} \right]. \tag{6}$$

2) Edge Computing Model: Once task$_{nm}$ is chosen to be offloaded to the ES, that is, $x_{nm}^{(1)} = 0$ and $x_{nm}^{(2)} = 1$. The task transmission time for task$_{nm}$ can be expressed as

$$T_{nm}^{\text{tran}} = \frac{w_{nm}}{b_n} \tag{7}$$

where b_n is the bandwidth between the nth MD and the ES.

When the task is offloaded to the ES or the CS, the offloaded program and data do not need to be returned to the MDs in the downlink, only the results are required. Thus, the response time and energy consumption in the downlink are much smaller than that of the uplink [31]. For simplicity, the response time and energy consumption in the downlink can be negligible. Therefore, the response time taken for task$_{nm}$ mainly includes the data transmission time and task execution time, which can be calculated by

$$T_{nm}^{\text{edge}} = \frac{\sigma w_{nm}}{f_e} + T_{nm}^{\text{tran}} \tag{8}$$

where f_e is denoted as the computational capacity (i.e., CPU cycles per second) of the ES.

The energy consumption when task$_{nm}$ is offloaded to the ES can be expressed as

$$E_{nm}^{\text{edge}} = \sigma \epsilon_e w_{nm} + e_t w_{nm} \tag{9}$$

where the energy consumed to transmit data of a unit size is e_t and the average energy required by the ES to process an instruction is ϵ_e.

As a consequence, the overall response time and energy consumed by the nth user on offloading tasks to the ES can be expressed as follows, respectively:

$$T_n^{\text{edge}} = \sum_{m=1}^{M} \left[\left(1 - x_{nm}^{(1)}\right) \cdot x_{nm}^{(2)} \cdot Te_{nm} \right] \tag{10}$$

$$E_n^{\text{edge}} = \sum_{m=1}^{M} \left[\left(1 - x_{nm}^{(1)}\right) \cdot x_{nm}^{(2)} \cdot E_{nm}^{\text{edge}} \right]. \tag{11}$$

3) Cloud Computing Model: Once task$_{nm}$ is selected to be offloaded to the CS, that is $x_{nm}^{(1)} = 0$ and $x_{nm}^{(2)} = 0$. Similarly, the response time and energy consumption when task$_{nm}$ is offloaded to the CS can be expressed as

$$T_{nm}^{\text{cloud}} = \frac{\sigma w_{nm}}{f_c} + \frac{w_{nm}}{b_n} \tag{12}$$

$$E_{nm}^{\text{cloud}} = \sigma \epsilon_c w_{nm} + e_t w_{nm} \tag{13}$$

where f_c denotes the computational capacity (i.e., CPU cycles per second) of the CS and ϵ_e denotes the average energy required by the CS to process an instruction.

Generally speaking, due to the elasticity of computing resources, CS has the strongest computational capacity, followed by ES, and MD is the weakest because of its constrained size. Therefore, we have $f_c > f_e > f_l$. In addition, due to differences in CPU architecture and cooling systems, the computational costs of EC and CS are much lower than that of MDs. What is more, due to the higher integration of central cloud equipment, its cost is even lower than that of edge cloud. Thus, we generally have $\epsilon_l > \epsilon_e > \epsilon_c$.

Let the total response time and total energy consumption of the nth user on offloading tasks to the CS be denoted as Tc_n and Ec_n, respectively, which can be expressed as

$$T_n^{\text{cloud}} = \sum_{m=1}^{M} \left[\left(1 - x_{nm}^{(1)}\right) \cdot \left(1 - x_{nm}^{(2)}\right) \cdot T_{nm}^{\text{cloud}} \right] \tag{14}$$

$$E_n^{\text{cloud}} = \sum_{m=1}^{M} \left[\left(1 - x_{nm}^{(1)}\right) \cdot \left(1 - x_{nm}^{(2)}\right) \cdot E_{nm}^{\text{cloud}} \right]. \tag{15}$$

B. Problem Formulation

Since the MDs do not affect the process of data transmission while computing, and the computing of the ES and the CS can also be carried out simultaneously, the overall response time taken by each user is the maximum of each MD, which can be expressed as

$$T^{\text{total}} = \sum_{n=1}^{N} \max\left\{ T_n^{\text{local}}, T_n^{\text{edge}}, T_n^{\text{cloud}} \right\}. \tag{16}$$

In addition, the overall energy consumption of the user is the sum of the energy consumed to process each task, that is

$$\begin{aligned} E^{\text{total}} &= \sum_{n=1}^{N}\left(E_n^{\text{local}} + E_n^{\text{edge}} + E_n^{\text{cloud}}\right) \\ &= \sum_{n=1}^{N}\sum_{m=1}^{M}\left[\left(x_{nm}^{(1)}E_{nm}^{\text{local}} + \left(1 - x_{nm}^{(1)}\right)x_{nm}^{(2)}E_{nm}^{\text{edge}}\right.\right. \\ &\quad \left.\left. + \left(1 - x_{nm}^{(1)}\right)\left(1 - x_{nm}^{(2)}\right)E_{nm}^{\text{cloud}}\right)\right]. \quad (17) \end{aligned}$$

According to the above definitions, for any task$_{nm}$, the weighted response time and energy consumption during the offloading process are closely related to the amount of data and the choice of offloading decisions, which can be formulated as

$$S(\boldsymbol{W}, \boldsymbol{X}) = \alpha E^{\text{total}} + (1 - \alpha)T^{\text{total}} \quad (18)$$

where $\boldsymbol{W} = \{w_{nm} | n \in \mathcal{N}; m \in \mathcal{M}\}$, $\boldsymbol{X} = \{x_{nm}^{(1)}, x_{nm}^{(2)} | n \in \mathcal{N}; m \in \mathcal{M}\}$, and $\alpha \in [0, 1]$ is a weighting coefficient to balance the importance of response time and energy consumption. For instance, when $\alpha > 0.5$, it indicates that energy consumption is more important than response time. Therefore, the optimal offloading decision-making problem can be transformed into an optimization problem $\mathcal{P}_1$

$$(\boldsymbol{\mathcal{P}_1}) \ \min_{\boldsymbol{X}} \ : \ S(\boldsymbol{W}, \boldsymbol{X}) = \alpha E^{\text{total}} + (1 - \alpha)T^{\text{total}} \quad (19)$$

$$\text{s.t.} : \ x_{nm}^{(1)}, x_{nm}^{(2)} \in \{0, 1\} \quad (20)$$

where the optimization problem $\mathcal{P}_1$ is a high-dimensional integer programming problem. It is easy to know that the number of possible offloading decisions is 3^{N*N}. When the number of MDs and the number of tasks increase, the feasible offloading decision state space grows exponentially, and as a result, heuristic decision algorithms will inevitably run slowly. Although the conventional optimization methods can theoretically obtain the globally optimal solution for the task offloading decision problem, it is difficult for them to provide the optimal offloading decision in a short time. In order to break the curse of high dimensionality and solve the problem of $\mathcal{P}_1$ efficiently, we develop a deep-learning-based approach for finding the optimal offloading decisions.

IV. MR-DRO ALGORITHM

In order to quickly and flexibly find the optimal offloading decision from a dynamic IoT environment, we design a novel MR-DRO algorithm, which aggregates the rapid environment learning ability of meta-RL, and the perception and decision-making ability of DRL.

A. MR-DRO Framework

Accordingly, the overall framework of the proposed MR-DRO algorithm can be divided into two parts, namely, the meta-RL model and the DRL model, as shown in Fig. 2.

Before making the offloading decision, MDs first provide information about tasks. At the same time, MDs collect offloading environment information to guide the decision-making process.

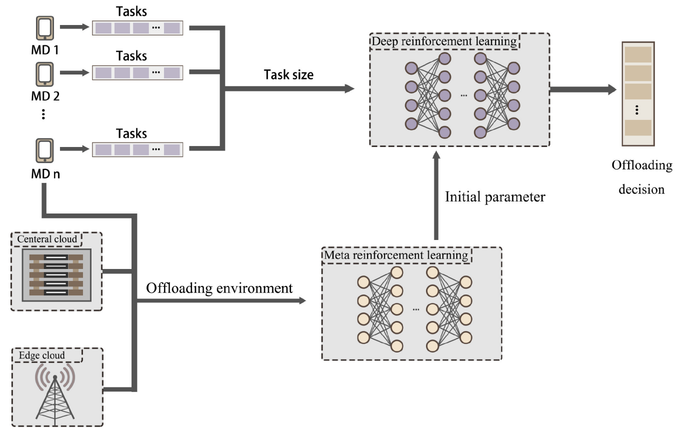


Fig. 2. Framework of the proposed MR-DRO algorithm.

Although the meta-RL model is not responsible for the decision-making process, it can generate appropriate initial training parameters in a relatively short time according to the existing training experience, thereby shortening the time required for training the DNN. In this framework, the meta-RL model reads the offloading environment information provided by MDs, determines the input layer, output layer, and other structures of DNNs, and gives the initial parameters of DNNs in the DRL model. Once the offloading environment information changes, e.g., network conditions, edge computing resources, and cloud computing resources, the meta-RL model can quickly provide appropriate initial training parameters and accelerate the training process of the DRL model. Therefore, the rationality of using meta-RL is to improve the generalization ability of the model.

In the case when the training samples are insufficient, DRL has a good performance in the application of multiclassification problems. In this framework, the DRL model reads the task information, initializes several parallel DNNs with the initial parameters provided by the meta-RL model, and then transforms the unsupervised learning process into a supervised learning process through the cyclic process of training and updating the data set. By doing this, we can improve the accuracy of the data set and update the parameters of DNNs and further provide a more accurate offloading decision. The specific algorithm flow and framework of the meta-RL model and DRL model will be further elaborated in the following sections.

B. Meta-RL Model

Different from the mainstream conventional machine learning algorithms, e.g., federated learning and reinforcement learning, the metadata set used by meta-RL is a series of metadata, which is also known as training tasks. Each training task contains the training set, test set, and training results during training. By learning a large number of training tasks, the learning ability of meta reinforcement neural network is continuously improved, so that when facing new tasks, it can complete the learning process faster and increase the training speed.

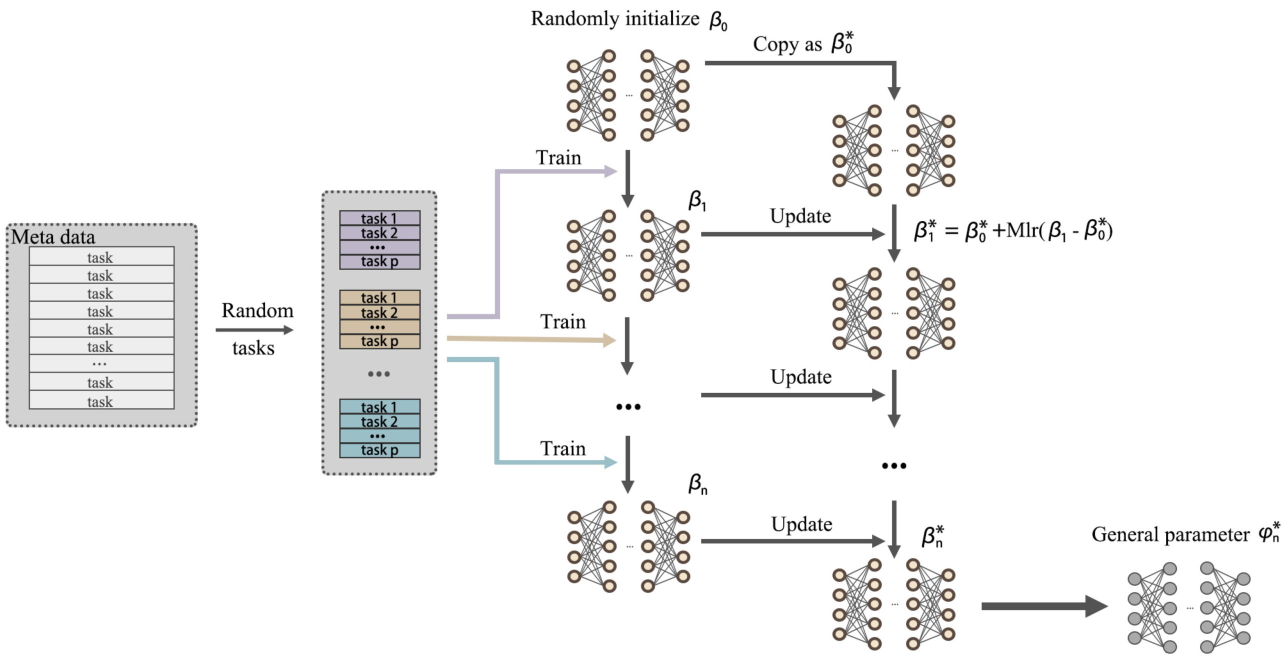


Fig. 3. Procedure of the meta-RL model.

Various types of meta-RL algorithms have been proposed, e.g., MAML, Reptile, and LSTM-based meta-learning algorithms. Although using MAML to train a meta-RL network can effectively reduce the training steps of a decision model, it involves the calculation of the second-order gradient. For large-scale issues, e.g., offloading decisions in heterogeneous edge/cloud computing environments, it will bring more computational costs, which severely affects the portability of the overall model and the level of offloading decision making. On the contrary, by increasing the training steps, the Reptile algorithm omits the process of calculating the second-order gradient and significantly reduces the training cost of the model. To the best of our knowledge, MR-DRO is the first work to formally adopt the Reptile algorithm for making offloading decisions in heterogeneous edge/cloud computing environments.

The specific process of the meta-RL model is shown in Fig. 3. First, according to the offloading environmental information provided by mobile terminal users, the neural network structure of the input and output layers of the meta-RL model can be determined. The weight parameters β_0 are randomly initialized and copied to record the starting point of training β_0^*. Then, we randomly select p pieces of metadata from the metadata set to form the training set and the new weight parameters β_1 are obtained after the training set is disturbed. We further calculate the difference $\beta_0 - \beta_1$ from that. Then, taking the difference value as the descending direction, the weight of neural network β_1^* for the learning rate Mlr can be updated as follows:

$$\beta_1^* = \beta_0^* + Mlr\big(\beta_1 - \beta_0^*\big). \tag{21}$$

Finally, repeat the above operations until the number of steps is reached. The parameters β_n^* obtained from the training can be used as the initial parameters φ_n^* of the DNN. We repeat the Reptile process K times according to the number of parallel DNNs in the DRL model.

Algorithm 1 Meta-RL-Based Algorithm

Input: Metadata
Output: Initial parameter φ
1: **for** $i = 1, 2, 3, \cdots, K$ **do**
2: Initialize the i^{th} DNN with random parameter β_0^i
3: Replicate the parameter as β_0^*
4: **for** $j = 1, 2, 3, \cdots, n$ **do**
5: Randomly choose a batch of tasks
6: Train the i^{th} DNN and update the parameter β_{j-1}^i as β_j^i
7: Calculate the meta parameter β_j^*
8: **end for**
9: Store β_n^* as initial parameter φ_i^*
10: **end for**
11: **return** Initial DNNs parameter φ

The algorithmic process of the proposed meta-RL algorithm is as described in Algorithm 1. First, we use the offloading environment information collected by MDs to decide the structure of each DNN. We randomly initialize the DNN (line 2). The DNN was trained for n steps using metadata, which is generated by a greedy algorithm (line 6). After training, we store the parameters of DNN as the initial parameters of the DRL model (line 9). Then, we repeat the whole process for K times to generate K different initial parameters.

C. DRL Model

Due to the particularity of edge computing and cloud computing, the training samples are generally rare or insufficient, which makes it difficult to apply the conventional machine learning algorithms. The DRL model can better solve the problem of data shortage for training, so it has become a

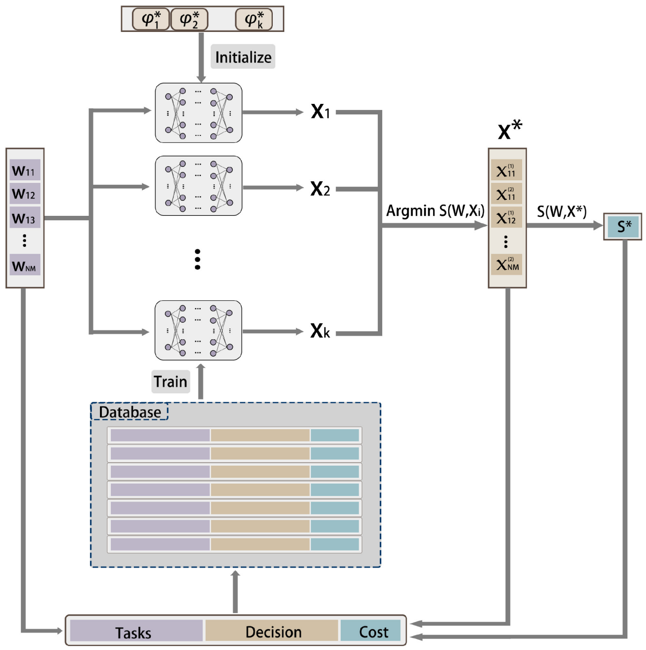


Fig. 4. Procedure of the DRL model.

common method in the field of edge computing. The procedure of the DRL model is demonstrated in Fig. 4.

1) Decision Generation: The DRL model contains K parallel DNNs as the core, the input of each DNN is the size information about the tasks and the output is the offloading decision of each task. We use a pair of decision indicators $\{x_{nm}^{(1)}, x_{nm}^{(2)}\}$ to represent the offloading scheme of task$_{nm}$. When initializing the model, DNN parameters $\Phi = \{\varphi_1, \varphi_2, \ldots, \varphi_k\}$ are first initialized according to the initialization parameter set $\Phi^* = \{\varphi_1^*, \varphi_2^*, \ldots, \varphi_n^*\}$ provided by the meta-RL model. These K DNNs have the same number of layers, nodes, and hyperparameter settings. However, due to the different initialization parameters, the weight parameters of each DNN are also different. Therefore, when faced with the same input, the outputs of these K DNNs are also different.

A group of task information $\boldsymbol{W}$ to be offloaded is generated randomly, and the size of each task should conform to the size distribution of tasks in real-world IoT environments. The task information of this group is input into K parallel DNNs for calculation, so that the K DNNs give their respective outputs, that is, K possible offloading schemes $\{\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots, \boldsymbol{X}_K\}$ are obtained. The offloading performance in terms of response time and energy consumption of each offloading scheme can be calculated by substituting each offloading scheme $\boldsymbol{X}_i$ into the cost function $S(\boldsymbol{W}, \boldsymbol{X})$. We compare the cost of all schemes and choose the offloading scheme $\boldsymbol{X}^*$ with the least total cost as the optimal one corresponding to this group of tasks.

Because the DNN has not been trained after initialization, there is still a certain gap between the decision given in the above way and the globally optimal decision. However, this decision is the one with the best performance among the K group decisions generated by the DNN. Thus, it can be known theoretically that if the sample composed of the task information and decisions of this group is used to train the other DNN groups, the updated weights should have a positive effect on reducing the total cost of the decision. Therefore, task information $\boldsymbol{W}$, decision information $\boldsymbol{X}^*$, and their corresponding total cost S^* are stored in the data set. Then, we set the number of samples in the data set and repeat the above process. Multiple groups of samples are randomly generated and stored in the data set until the upper limit of the data set is reached, which can be used as the training samples for initial training.

2) Model Training: After the data set is generated, K parallel DNNS are trained. Since all the DNNs share the same data set, Q samples are randomly selected from the data set as the training set during the training process of each neural network, and the order of the training set is disturbed to train the DNN.

To determine the reward value of the DRL model, we input the task information of the training set into K DNNs and generate offloading decisions for each group of tasks. Then, we can derive the cost of these decisions with the help of the cost function $S(\boldsymbol{W}, \boldsymbol{X})$. Then, the reward value can be calculated through the difference between the newly derived cost and the cost in the training set.

Since the output of DNNs is not always an integer, and the decision indicator is a parameter with a value of 0 or 1, the mean square error (MSE) is adopted to define the distance between the output of DNNs and the offloading scheme. The MSE formula can be expressed as

$$\mathrm{MSE} = \frac{1}{N}\sum_{t=1}^{N} ||\mathrm{logits}_t - \mathrm{outputs}_t)||^2. \tag{22}$$

For the results output from the output layers of DNNs, we take the offloading scheme closest to it as the offloading scheme of its output. Because each decision parameter can only take a value of 0 or 1, it is easy to know the output of each output layer node through the definition of MSE. If $\mathrm{outputs}^* > 1/2$, then $\mathrm{logits}^* = 1$, if $\mathrm{outputs}^* < 1/2$, then $\mathrm{logits}^* = 0$.

We adopt the cross-entropy expression as the loss function of the neural network. According to the loss function, the gap between decisions generated by DNNs and decisions given by the data can be calculated. We can use it to update the parameters of DNNs. The cross-entropy is minimized by the method based on gradient descent, which can be specifically expressed as

$$L(\varphi_i) = -\boldsymbol{X}^T \log f_{\varphi_i}(\boldsymbol{W}) - (1 - \boldsymbol{X})^T \log\big(1 - f_{\varphi_i}(\boldsymbol{W})\big) \tag{23}$$

where φ_i is the parameter of the ith DNN, and f_{φ_i} is its parameter expression. After training the DNNs in this way, the decision-making level is improved. In addition, the model can generate new samples according to the sample generation method described in the previous part, and update part of the old samples in the original data set with the new samples to obtain a more accurate data set. Using this method, we can continuously improve the accuracy of the data set and improve the decision-making level of the DNNs.

The algorithmic process of the proposed DRL algorithm [5] is as described in Algorithm 2. First, we initialize K DNNs

Algorithm 2 DRL-Based Dynamic Offloading Algorithm

Input: Workloads $\boldsymbol{W}$
Output: Optimal offloading decisions
1: **Initialization:** Initialize K DNNs with the parameter set Φ; Empty *database*
2: **for** $j = 1, 2, 3, \cdots, N$ **do**
3: Randomly generate a group of task information $\boldsymbol{W}_i$
4: **for** $i = 1, 2, 3, \cdots, K$ **do**
5: Replicate the information $\boldsymbol{W}_i$ to the i^{th} DNN
6: Generate the i^{th} offloading decision candidate $\boldsymbol{X}_i$ from the i^{th} DNN
7: **end for**
8: Select offloading decision $\boldsymbol{X}_i^*$ by minimizing $S(\boldsymbol{W}_i, \boldsymbol{X}_i)$
9: Calculate $S(\boldsymbol{W}_i, \boldsymbol{X}_i^*)$ as S^*
10: **if** *database* is not full **then**
11: Store $(\boldsymbol{W}_i, \boldsymbol{X}_i^*, S^*)$ into *database*
12: **else**
13: Discard the oldest sample and save the new one
14: Randomly choose K batches of samples from *database*
15: Train each DNN using a selected batch
16: **end if**
17: **end for**
18: **for** $i = 1, 2, 3, \cdots, K$ **do**
19: Replicate the information $\boldsymbol{W}$ to the i^{th} DNN
20: Generate the i^{th} offloading decision candidate $\boldsymbol{X}_i$ from the i^{th} DNN
21: **end for**
22: Select offloading decision $\boldsymbol{X}_i^*$ by minimizing $S(\boldsymbol{W}, \boldsymbol{X}_i)$
23: **return** Optimal offloading decisions $\boldsymbol{X}_i^*$

with the parameter set generated by the meta-RL model (line 1). We train the model for N steps. During each step, if the database is not full, we store the newly generated sample as train data (line 10). If the database is full, we train each DNN with a batch of samples randomly selected from the database. Then, we use the new sample to replace the oldest sample to increase the accuracy of the database (line 12). After the model is trained, we replicate the workload to each DNN and generate several offloading-decision candidates (line 20). Then, we output the decision with the best performance as the final decision. If the workload is changed, we do not need to train the whole model again. It can still solve the problem properly.

D. Testing

First, to verify that K parallel DNNs in the DRL model will converge after finite training steps, we need to prove that the decision level of each DNN basically remains unchanged. We define Q_1 as the convergence rate of the model, which can be expressed as

$$Q_1 = \frac{1}{q}\sum_{j=1}^{q}\frac{\min\left(S_j, S_j^*\right)}{\max\left(S_j, S_j^*\right)} \tag{24}$$

where q is the number of samples contained in the data set acquired for training each DNN, S_j^* is the total cost of the offloading scheme recorded in the jth sample, and S_j is the new total cost of the optimal offloading decision obtained from the model. When the decision level of the DRL model is basically unchanged, the total cost before and after training should be basically the same. In other words, if $([\min(S_j, S_j^*)]/[\max(S_j, S_j^*)])$ is closer to 1, we can say that the model converges much better. Thus, the convergence performance of the model can be known by the values of Q_1 during each training.

In addition, the convergence of the model does not guarantee its decision-making accuracy. The model itself may converge to a locally optimal solution in the case when the weight parameters remain unchanged. In order to intuitively measure the gap between the offloading decision given by the model and the globally optimal offloading decision, we randomly generate task groups and calculate the corresponding globally optimal offloading decision of each task group by means of traversal. Then, the minimum total cost is calculated, and the r groups of samples from a new data set called the standard set, which is used to test the offloading decision level of the model. We define Q_2 as the accuracy rate of the model, which is derived as

$$Q_2 = \frac{1}{r}\sum_{t=1}^{r}\frac{S_t^{\text{optimal}}}{S_t^{\text{test}}} \tag{25}$$

where S_t^{optimal} and S_t^{test} are the total costs of the globally optimal offloading decision of the tth group of tasks, and the offloading decision given by the model, respectively. It is easy to know that $S_t^{\text{test}} \geq S_t^{\text{optimal}}$. If the offloading decision given by the model is closer to the globally optimal offloading decision, then Q_2 is closer to 1. Therefore, when the model is trained to convergence, the offloading decision accuracy of the model can be known only by calculating the value of Q_2.

Moreover, the decision making is also related to the hyper-parameters of the model itself, e.g., the number of parallel DNNs and the learning rate of the model. In addition, the size of the data set also affects the convergence of the model.

V. Performance Evaluation

In this section, we evaluate the effectiveness of the proposed MR-DRO algorithm through numerical simulations under different edge computing scenarios. Besides, we also specifically compare the proposed scheme with several different offloading strategies.

A. Parameter Settings

Considering a heterogeneous edge/cloud environment consisting of three MDs, one ES, and one CS, where each MD has three tasks. We assume that the size of each task is randomly distributed between 10 and 30 MB. Additionally, we assume that the CPU needs to execute 1000 instructions when calculating each unit of the task, the CPUs in MDs, ES, and CS consume 3.0, 1.5, and 1.0 mJ to run each instruction. Also, we set the energy consumed to transmit a unit of data is 0.1 mJ. Referring to the computing capacities in the actual situation, we set the clock frequencies of MDs, ES, and CS as $f_l = 100$ MHz, $f_e = 600$ MHz, and $f_c = 1000$ MHz, respectively. Besides, we set the weighting parameter $\alpha = 0.5$, which means that the response time is as important as energy consumption. Through pretraining and comparison of existing research results, we set each DNN in the DRL model

TABLE III
EVALUATION PARAMETER

Evaluation Parameters	**Values**
The number of instructions to run per unit of task	σ =1000
The energy to transmit a unit of data	$e_t = 0.1$ mJ
The number of MDs	$N = 3$
The number of tasks	$M = 3$
The amount of data for each task ω_{nm}	$10 \sim 30$ MB
The network bandwidth of the n_{th} MD	$b_n = 500$ Mbps
The processing rate of the MDs	$f_l = 100$ MHz
The processing rate of the ES	$f_e = 600$ MHz
The processing rate of the CS	$f_c = 1000$ MHz
The energy to execute each instruction on the MDs	$\epsilon_l = 3.0$ mJ
The energy to execute each instruction on the ES	$\epsilon_l = 1.5$ mJ
The energy to execute each instruction on the CS	$\epsilon_l = 1.0$ mJ

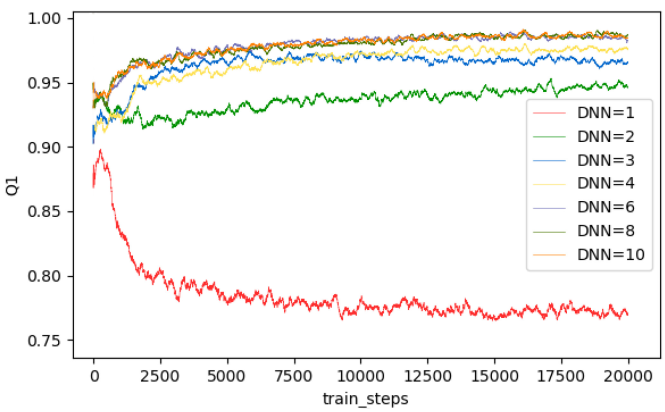


Fig. 5. Impact of the number of DNNs.

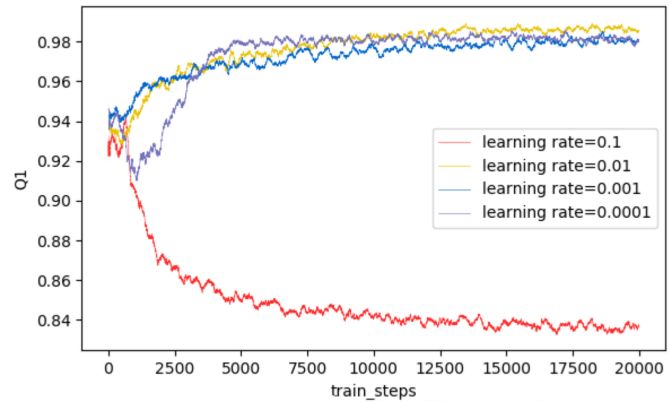


Fig. 6. Impact of the learning rate.

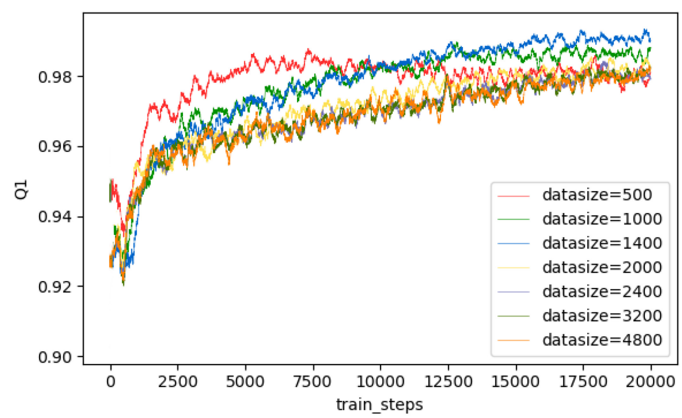


Fig. 7. Impact of the size of database.

to include two hidden layers. The details of our parameter settings are shown in Table III.

B. Convergence Performance

In this part, the convergence performance of MR-DRO is first illustrated. We will separately analyze the impact of different numbers of DNNs, sizes of the database, and learning rates on convergence performance.

1) Impact of Number of DNNs: We adjust the number of DNNs K from 1 to 10 and train each model for 20 000 steps. During the training of each neural network, we randomly select 128 samples from the data set as the training set. As shown in Fig. 5, it can be seen that when $K = 1$, the model cannot converge through the training process since its Q_1 value does not converge to 1 as the training steps increases. As the number of DNNs increases, the convergence performance of our model will be improved. Considering the energy consumption and response time spent during the training model, we choose $N = 8$ as a compromise between total consumption and convergence performance.

2) Impact of Learning Rate: The learning rate is also an essential hyperparameter that affects the decision-making model. If the learning rate is too large, the neural network will get more exploration results and it is more difficult to reach the convergence, so that the optimal solution cannot be accurately obtained. On the contrary, when the learning rate is too small, the convergence speed will be slowed down to a certain extent and the neural network may treat the local optimal solution as the globally optimal solution, that is, it is easier to fall into the local optimal. From Fig. 6, we can find that the model achieves the best performance when the learning rate is 0.01.

3) Impact of Size of Database: The size of the database also affects the performance of the model. When the size of the database is larger, the old samples in the database cannot be replaced with newly generated and more accurate samples in time; on the contrary, when the sale of the database is smaller, the DNN cannot be trained well. As depicted in Fig. 7, we test different sizes of database and find out that the model achieves the best performance when it equals 1400. In addition, it can be seen that MR-MRO improves the convergence speed even when the database is small in scale.

C. Accuracy Performance

Similar to the model convergence rate, the model accuracy rate is also a critical indicator for measuring the offloading ability of the algorithm. When the model converges, it can only indicate that the parameters of the model have reached a relatively stable state after training. Even if the training step length is extended, the offloading decision will not change greatly. However, the decision may not be the globally optimal solution for the offloading scheme. As a result, we need to calculate the value of Q_2 to accurately represent the specific decision-making level of the algorithm.

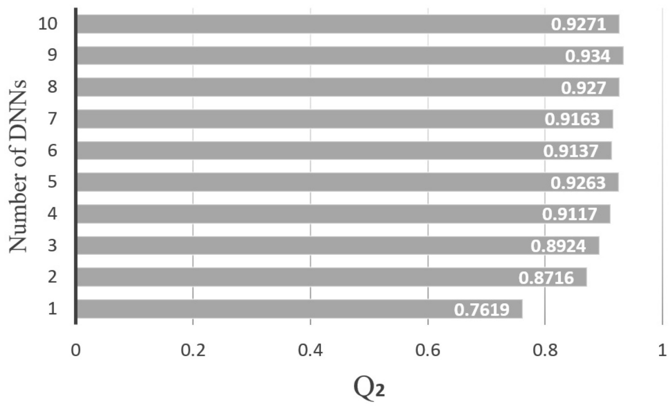


Fig. 8. Impact of the number of DNNs on Q_2.

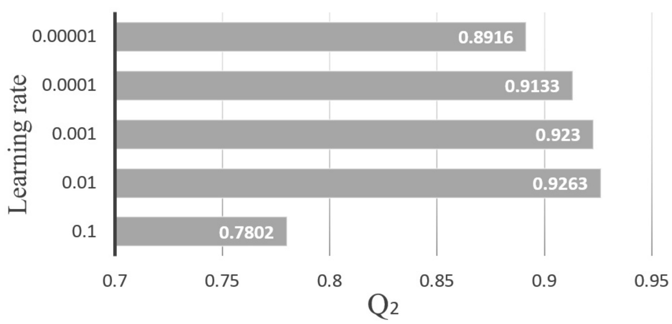


Fig. 9. Impact of the learning rate on Q_2.

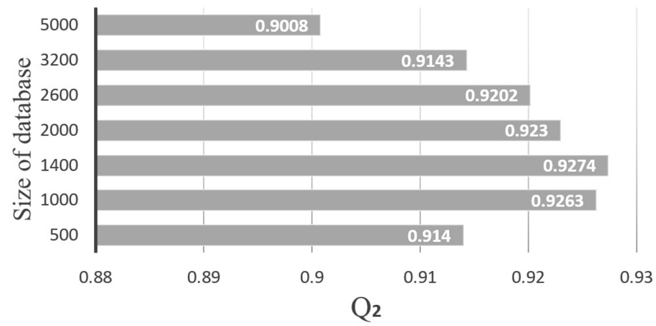


Fig. 10. Impact of the size of database on Q_2.

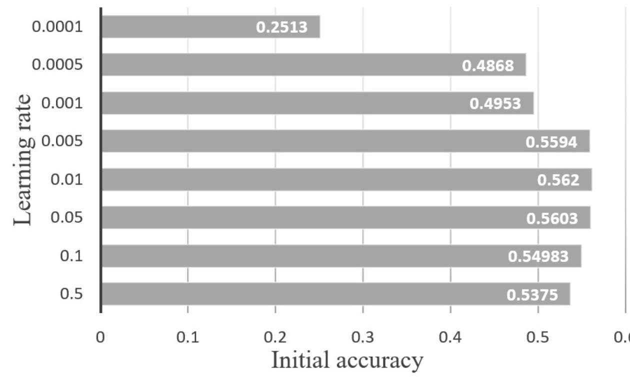


Fig. 11. Initial accuracy under different learning rates.

To calculate the value of Q_2, we randomly generate 512 groups of tasks and calculate the globally optimal offloading decision for each sample through the cost function $S(\boldsymbol{W}, \boldsymbol{X})$. Then we get a test data set, which is known as the standard set. When the model reaches convergence after 20 000 steps of training, we input the sample of each group of tasks into our model, and generate the offloading decision, respectively. Using the optimal offloading decisions and the offloading decisions generated by our algorithm, we calculate the value of Q_2. By comparing the changes of Q_2 value under different variables, the following conclusions can be drawn.

1) Impact of Number of DNNs: As shown in Fig. 8, when the number of DNNs increases, the accuracy of the model will also increase. When $K > 7$, the accuracy of the model reaches the highest point, and its Q_2 value basically remains at 0.94, which means that the error between it and the globally optimal solution is about 5%.

2) Impact of Learning Rate: It can be seen from Fig. 9 that when the learning rate is 0.1, the model accuracy is relatively low, whose Q_2 value is less than 0.8. In particular, when the learning rate is 0.01, the accuracy of the model is significantly improved. In addition, considering that a low learning rate will reduce the training speed of DNNS and cause unnecessary costs, the optimal learning rate is chosen as 0.01.

3) Impact of Size of Database: It can be seen from Fig. 10 that the size of the database has a relatively small impact on the model. When the size of the database is between 500 and 4800, the accuracy of the model is higher than 0.9. In particular, when the number of samples is 1000–1400, the model has the best accuracy performance.

D. Meta-RL Performance

In this section, we specifically discuss the performance of meta-RL models. First, we generate a metadata set based on a greedy algorithm. The metadata set contains 10 000 metadata, where 50 metadata are randomly selected as the training set for each step of training. We set the default training step length to 2000. Then, we use the same method to generate a standard set, which is composed of 512 groups of samples. After setting the learning rate of meta-RL to different values, we test the initial accuracy Q_2 of the DRL model initialized by meta-RL without training. As depicted in Fig. 11, when the learning rate is set to 0.01–0.05, the initial accuracy performance is the best.

In the above experiment, we set the learning rate to 0.01, and only change the number of DNNs in the model. Before performing the meta-RL algorithm, we first randomly initialize the neural network parameters and check the value of Q_2 as the initial training accuracy. After that, every 500 steps of training, we test the value of Q_2 of the model parameters.

Since the initialization is random and the training samples are randomly selected, the accuracy of the model will fluctuate when the training steps are not enough. Therefore, we take the average of the accuracy of multiple tests as the final result. As shown in Fig. 12, when the meta-RL model is not trained, the accuracy of the randomly initialized model is low, especially when the number of DNNs is small, the accuracy is even as

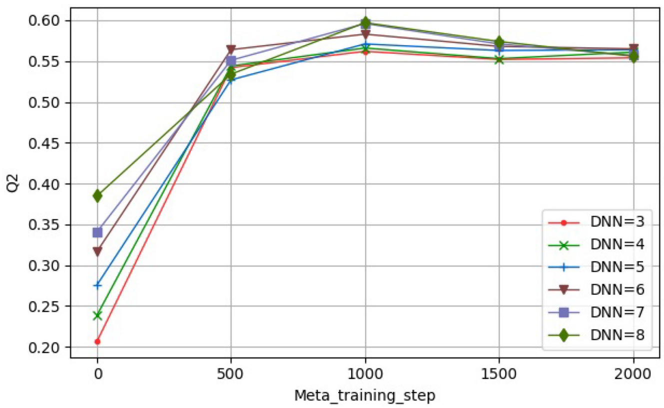


Fig. 12. Impact of the number of DNNs.

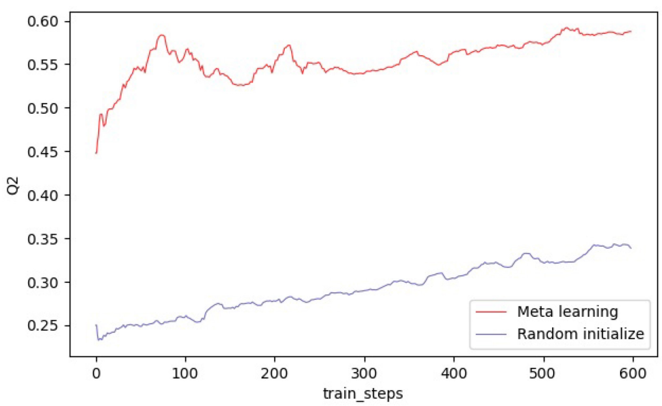


Fig. 13. Comparison with different initialization methods.

low as about 20%. Using a few hundred steps of meta-RL, we can greatly improve the initial decision accuracy of the model to more than 50% and speed up the training process of the DRL model to a large extent.

After that, we initialize the DRL model through the meta-RL model and random initialization, respectively. Then, we do the subsequent training steps, and after each step of training, we test the value of Q_2. It can be observed from Fig. 13 that when the decision-making model does not go through the meta-RL model, random initialization will lead to low initial accuracy, thereby more rounds of training are required. After meta-RL, the initial parameters provided by it are used to initialize the DRL model, and its initial accuracy is greatly improved. Therefore, the steps of subsequent training can be greatly reduced and the portability of the model can be improved. At the same time, by separately calculating the time used by the meta-RL model and the DRL model, it can be seen that the training process of the meta-RL model is extremely short. Therefore, the meta-RL model will not bring high training costs to the whole system.

E. Performance Comparison

1) *Local-Only No Offloading Scheme:* In this method, each mobile user chooses to execute its task locally on the MD.

TABLE IV
COMPARISON OF DIFFERENT SCHEMES

Offloading schemes	Total energy consumption	Q_2 value
Local-only	2900 J	0.14
ES-only	564 J	0.68
CS-only	580 J	0.70
Local & ES	550 J	0.74
Local & CS	520 J	0.78
Genetic	468 J	0.87
MR-DRO	432 J	0.94
Exhaustive search	408 J	1.00

2) *ES-Only Full Offloading Scheme:* In this method, all computing tasks are fully offloaded to the ES for execution.
3) *CS-Only Full Offloading Scheme:* In this method, all computing tasks are fully offloaded to the CS for execution.
4) *Local and ES Partial Offloading Scheme:* In this method, some tasks are processed locally on the MDs, while some of them are offloaded to the ES for execution.
5) *Local and CS Partial Offloading Scheme:* In this method, some tasks are processed locally on the MDs, while some of them are offloaded to the CS for execution.
6) *Genetic Partial Offloading Scheme [18]:* In this method, a GA is adopted for finding near-optimal offloading decisions over the MDs, the ES, and the CS.
7) *MR-DRO Partial Offloading Scheme:* In this method, we apply the proposed MR-DRO algorithm to generate near-optimal offloading decisions over the MDs, the ES, and the CS.
8) *Exhaustive Search Scheme:* We exhaustively search the optimal one among all feasible offloading decisions over the MDs, the ES, and the CS.

Based on the above discussion, we set the number of DNNs as 8, the size of the database as 1400, and the learning rate as 0.01. In the meanwhile, in order to intuitively express the practical significance of this method, we calculate the globally optimal offloading decision for each group of samples under several offloading schemes. Then, we calculate the total consumption caused by each offloading scheme, by which we can derive the value of Q_2.

As depicted in Table IV, the proposed *MR-DRO* scheme outperforms other offloading-decision approaches significantly. For example, the Q_2 value of the *Local-only* scheme is only about 0.14 since it has to process a large number of compute-intensive tasks locally on MDs, while the Q_2 value of the *MR-DRO* scheme approaches one. This is because unlike the *Local-only*, *ES-only*, and *CS-only* schemes, the *MR-DRO* scheme dynamically offloads tasks according to the heterogeneous edge/cloud computing environment. Compared to the *exhaustive search* scheme, the designed *MR-DRO* scheme is able to obtain sufficiently accurate for maximizing the offloading performance, without a huge computation cost. Compared to the Local-only scheme, our scheme reduces the total consumption by 85.1%. Also, this method can reduce the total consumption by 22% on average compared with other methods.

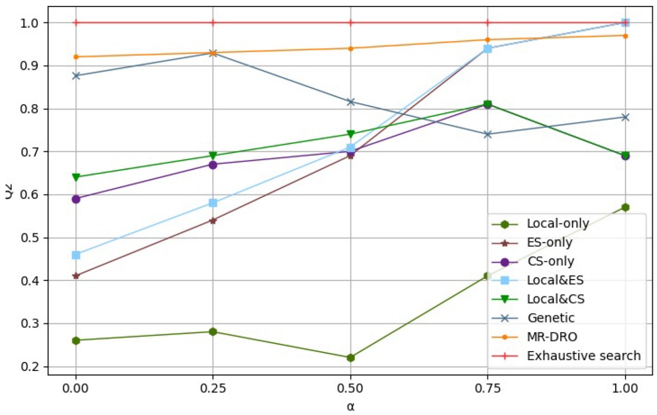


Fig. 14. Comparison with several offloading schemes under different weighting parameters.

Furthermore, in order to verify the performance of the proposed MR-DRO algorithm, we use a GA to generate offloading decisions in the same offloading scenario. The GA algorithm generally involves multiple steps, such as encoding, fitness functions, initialization and selection, crossover and mutation, and local search, which will affect the efficiency of problem solving [32]. From Table IV and Fig. 14, the simulation results demonstrate that our MR-DRO algorithm is more reliable and achieves superior performance under different weighting parameters. In addition, the computational complexity of GA varies with model complexity, which is not suitable for large-scale edge/cloud computing scenarios.

In order to further ensure the decision-making level of our algorithm, we adjust the weighting parameter α. As shown in Fig. 14, regardless of the different importance of response time and energy consumption that mobile users are concerned about, the *MR-DRO* scheme can always achieve the best offloading performance in terms of response time and energy consumption. Therefore, it can achieve near-optimal offloading decisions in edge and cloud computing heterogeneous environments.

VI. Conclusion and Future Work

In this article, we have proposed the MR-DRO algorithm to obtain near-optimal offloading decisions in a heterogeneous edge/cloud computing environment. The MR-DRO framework includes a parameter-initialing model based on meta-RL, and a decision-making model based on DRL. The former generates the initial parameters for training, improves the accuracy of the decision-making model, and greatly increases the portability of the model. It improves the performance of the algorithm when handling sophisticated offloading scenarios by adopting the Reptile algorithm. The latter one applies multiple parallel DNNs to determine when and where each task should be offloaded, and the offloading performance in terms of response time and energy consumption is significantly improved compared with many baseline methods.

Even though this study only considered a simple offloading scenario with only one ES and one CS, MR-DRO can be easily expanded and generates offloading decisions for complicated real-world scenarios with multiple ESs or multiple clouds. In the meantime, when the total bandwidth is fixed and allocatable, the optimal bandwidth allocation problem can also be treated as a convex problem and solved easily [33]. In future work, we intend to conduct preliminary meta-reinforcement on the network parameters of DNNs in a variety of ways under various constraints and feed the meta-RL model directly into the decision generation model to further improve the portability of the model.

References

[1] 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.

[2] 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.

[3] N. Muslim, S. Islam, and J.-C. Grégoire, "Offloading framework for computation service in the edge cloud and core cloud: A case study for face recognition," *Int. J. Netw. Manag.*, vol. 31, no. 4, Nov. 2020, Art. no. e2146.

[4] G. Peng, H. Wu, H. Wu, and K. Wolter, "Constrained multiobjective optimization for IoT-enabled computation offloading in collaborative edge and cloud computing," *IEEE Internet Things J.*, vol. 8, no. 17, pp. 13723–13736, Sep. 2021.

[5] H. Wu, Z. Zhang, C. Guan, K. Wolter, and M. Xu, "Collaborate edge and cloud computing with distributed deep learning for smart city Internet of Things," *IEEE Internet Things J.*, vol. 7, no. 9, pp. 8099–8110, Sep. 2020.

[6] S. Yu, X. Wang, and R. Langar, "Computation offloading for mobile edge computing: A deep learning approach," in *Proc. IEEE 28th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, 2017, pp. 1–6.

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

[8] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan, and X. Chen, "Convergence of edge computing and deep learning: A comprehensive survey," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 2, pp. 869–904, 2nd Quart., 2020.

[9] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile edge computing," *IEEE Internet Things J.*, early access, Jun. 22, 2021, doi: 10.1109/JIOT.2021.3091142.

[10] L. Huang, L. Zhang, S. Yang, L. P. Qian, and Y. Wu, "Meta-learning based dynamic computation task offloading for mobile edge computing networks," *IEEE Commun. Lett.*, vol. 25, no. 5, pp. 1568–1572, May 2021.

[11] P. Shu *et al.*, "eTime: Energy-efficient transmission between cloud and mobile devices," in *Proc. IEEE INFOCOM*, Apr. 2013, pp. 195–199.

[12] Y. Li, S. Xia, M. Zheng, B. Cao, and Q. Liu, "Lyapunov optimization based trade-off policy for mobile cloud offloading in heterogeneous wireless networks," *IEEE Trans. Cloud Comput.*, early access, Aug. 30, 2019, doi: 10.1109/TCC.2019.2938504.

[13] E. E. Haber, T. M. Nguyen, D. Ebrahimi, and C. Assi, "Computational cost and energy efficient task offloading in hierarchical edge-clouds," in *Proc. IEEE 29th Annu. Int. Symp. Pers. Indoor Mobile Radio Commun. (PIMRC)*, Sep. 2018, pp. 1–6.

[14] S. Yu, B. Dab, Z. Movahedi, R. Langar, and L. Wang, "A socially-aware hybrid computation offloading framework for multi-access edge computing," *IEEE Trans. Mobile Comput.*, vol. 19, no. 6, pp. 1247–1259, Jun. 2020.

[15] H. Wu, W. J. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 7, pp. 1464–1480, Jul. 2019.

[16] W. Zhang and Y. Wen, "Energy-efficient task execution for application as a general topology in mobile cloud computing," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 708–719, Jul.–Sep. 2018.

[17] V. Haghighi and N. S. Moayedian, "An offloading strategy in mobile cloud computing considering energy and delay constraints," *IEEE Access*, vol. 6, pp. 11849–11861, 2018.

[18] X. Xu *et al.*, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Gener. Comput. Syst.*, vol. 95, pp. 522–533, Jun. 2019.

[19] M. Li, Q. Wu, J. Zhu, R. Zheng, and M. Zhang, "A computing offloading game for mobile devices and edge cloud servers," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–10, Dec. 2018.

[20] K. R. Alasmari, R. C. Green, and M. Alam, "Mobile edge offloading using Markov decision processes," in *Edge Computing (EDGE)*. Cham, Switzerland: Springer, 2018, pp. 80–90.

[21] M. B. Terefe, H. Lee, N. Heo, G. C. Fox, and S. Oh, "Energy-efficient multisite offloading policy using Markov decision process for mobile cloud computing," *Pervasive Mobile Comput.*, vol. 27, pp. 75–89, Apr. 2016.

[22] 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.

[23] H. Li, K. Ota, and M. Dong, "Learning IoT in edge: Deep learning for the Internet of Things with edge computing," *IEEE Netw.*, vol. 32, no. 1, pp. 96–101, Jan./Feb. 2018.

[24] Y. Kang *et al.*, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *SIGARCH Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, Apr. 2017.

[25] B. Dab, N. Aitsaadi, and R. Langar, "*Q*-learning algorithm for joint computation offloading and resource allocation in edge cloud," in *Proc. IFIP/IEEE Symp. Integr. Netw. Service Manag. (IM)*, 2019, pp. 45–52.

[26] S. Yu, X. Chen, L. Yang, D. Wu, M. Bennis, and J. Zhang, "Intelligent edge: Leveraging deep imitation learning for mobile edge computation offloading," *IEEE Wireless Commun.*, vol. 27, no. 1, pp. 92–99, Feb. 2020.

[27] L. Huang, X. Feng, A. Feng, Y. Huang, and L. P. Qian, "Distributed deep learning-based offloading for mobile edge computing networks," *Mobile Netw. Appl.*, vol. 2018, pp. 1–8, Nov. 2018.

[28] G. Qu, H. Wu, R. Li, and P. Jiao, "DMRO: A deep meta reinforcement learning-based task offloading framework for edge-cloud computing," *IEEE Trans. Netw. Service Manag.*, vol. 18, no. 3, pp. 3448–3459, Sep. 2021.

[29] J. Wang, J. Hu, G. Min, A. Y. Zomaya, and N. Georgalas, "Fast adaptive task offloading in edge computing based on meta reinforcement learning," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 1, pp. 242–253, Jan. 2021.

[30] M. Mukherjee, M. Guo, J. Lloret, and Q. Zhang, "Leveraging intelligent computation offloading with fog/edge computing for tactile Internet: Advantages and limitations," *IEEE Netw.*, vol. 34, no. 5, pp. 322–329, Sep./Oct. 2020.

[31] Q. Luo, C. Li, T. Luan, and W. Shi, "Minimizing the delay and cost of computation offloading for vehicular edge computing," *IEEE Trans. Services Comput.*, early access, Mar. 9, 2021, doi: 10.1109/TSC.2021.3064579.

[32] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent IoT applications in edge and fog computing environments," *IEEE Trans. Mobile Comput.*, vol. 20, no. 4, pp. 1298–1311, Apr. 2021.

[33] H. Zhang, H. Liu, J. Cheng, and V. C. M. Leung, "Downlink energy efficiency of power allocation and wireless backhaul bandwidth allocation in heterogeneous small cell networks," *IEEE Trans. Commun.*, vol. 66, no. 4, pp. 1705–1716, Apr. 2018.

Ziru Zhang received the bachelor's degree in mathematics and applied mathematics from Tianjin University, Tianjin, China, in 2021.

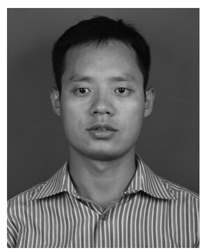
His research interests include machine learning and edge computing.

Nianfu Wang received the bachelor's degree in mathematics and applied mathematics from Tianjin University, Tianjin, China, in 2021. He is currently pursuing the master's degree with the School of Mathematics, Harbin Institute of Technology, Harbin, China.

His research interests include distributed deep learning and edge computing.

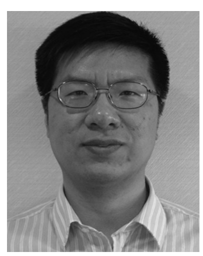
Huaming Wu (Member, IEEE) received the B.E. and M.S. degrees in electrical engineering from Harbin Institute of Technology, Harbin, China, in 2009 and 2011, respectively, and the Ph.D. degree (Highest Hons.) in computer science from Freie Universität Berlin, Berlin, Germany, in 2015.

He is currently an Associate Professor with the Center for Applied Mathematics, Tianjin University, Tianjin, China. His research interests include model-based evaluation, wireless and mobile network systems, mobile cloud computing, and deep learning.



Chaogang Tang (Member, IEEE) received the B.S. degree from Nanjing University of Aeronautics and Astronautics, Nanjing, China, in 2007, and the Ph.D. degree from the School of Information Science and Technology, University of Science and Technology of China, Hefei, China, and the Department of Computer Science, City University of Hong Kong, under a joint Ph.D. Program, in 2012.

He is currently with China University of Mining and Technology, Xuzhou, China. His research interests include mobile cloud computing, fog computing, Internet of Things, and big data.



Ruidong Li (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees in computer science from the University of Tsukuba, Tsukuba, Japan, in 2005 and 2008, respectively.

He is an Associate Professor with Kanazawa University, Kanazawa, Japan. Before joining this university, he was a Senior Researcher with the National Institute of Information and Communications Technology, Tokyo, Japan. His research interests include future networks, big data, intelligent Internet edge, Internet of Things, network security, information-centric network, artificial intelligence, quantum Internet, cyber–physical system, and wireless networks.

Dr. Li served as the Chair for several conferences and workshops, such as the General Co-Chair for IEEE MSN 2021, AIVR2019, and IEEE INFOCOM 2019/2020/2021 ICCN Workshop, and the TPC Co-Chair for IWQoS 2021, IEEE MSN 2020, BRAINS 2020, IEEE ICDCS 2019/2020 NMIC workshop, and ICCSSE 2019. He is the Secretary of IEEE ComSoc Internet Technical Committee (ITC) and the Founder and Chair of IEEE SIG on Big Data Intelligent Networking and IEEE SIG on Intelligent Internet Edge. He is an Associate Editor of IEEE INTERNET OF THINGS JOURNAL, and also served as the Guest Editor for a set of prestigious magazines, transactions, and journals, such as *IEEE Communications Magazine*, IEEE NETWORK, and IEEE TRANSACTIONS ON NETWORK SCIENCE AND ENGINEERING. He is a member of IEICE.