DMRO: A Deep Meta Reinforcement Learning-based Task Offloading Framework for Edge-Cloud Computing

Guanjin Qu, Huaming Wu, *Member, IEEE*, Ruidong Li, *Senior Member, IEEE*, and Pengfei Jiao

***Abstract*—With the explosive growth of mobile data and the unprecedented demand for computing power, resource-constrained edge devices cannot effectively meet the requirements of Internet of Things (IoT) applications and Deep Neural Network (DNN) computing. As a distributed computing paradigm, edge offloading that migrates complex tasks from IoT devices to edge-cloud servers can break through the resource limitation of IoT devices, reduce the computing burden and improve the efficiency of task processing. However, the problem of optimal offloading decision-making is NP-hard, traditional optimization methods are difficult to achieve results efficiently. Besides, there are still some shortcomings in existing deep learning methods, e.g., the slow learning speed and the weak adaptability to new environments. To tackle these challenges, we propose a Deep Meta Reinforcement Learning-based Offloading (DMRO) algorithm, which combines multiple parallel DNNs with Q-learning to make fine-grained offloading decisions. By aggregating the perceptive ability of deep learning, the decision-making ability of reinforcement learning, and the rapid environment learning ability of meta-learning, it is possible to quickly and flexibly obtain the optimal offloading strategy from a dynamic environment. We evaluate the effectiveness of DMRO through several simulation experiments, which demonstrate that when compared with traditional Deep Reinforcement Learning (DRL) algorithms, the offloading effect of DMRO can be improved by 17.6%. In addition, the model has strong portability when making real-time offloading decisions, and can fast adapt to a new MEC task environment.**

***Index Terms*—Internet of Things, Edge Computing, Task Offloading, Deep Neural Network, Meta Reinforcement Learning**

I. Introduction

With the rapid development of Internet of Things (IoT) and communication technologies, a large number of computation-intensive tasks need to be transferred from IoT devices to the cloud server for execution [1]. However, the task offloading process usually involves large amounts of data transmission, which will result in high latency for IoT applications. The emergence of Mobile Edge Computing (MEC) can effectively alleviate this challenge. As a distributed computing paradigm, edge offloading that migrates complex tasks from IoT devices to edge-cloud servers can provide computing services for edge caching, edge training, and edge inference [2]. Before the IoT application being offloaded to the cloud server, it needs to pass through the edge server, such as the base station. The edge server is closer to the device than the cloud server, so it has greater bandwidth and response time. By utilizing the computing and decision-making capabilities of the edge server, the task computing of the device can be offloaded to different servers, thereby reducing computing latency and energy consumption [3].

The process of task offloading is generally affected by a variety of factors in different areas, e.g., user preferences, wireless communication channels, network connection quality, mobility of IoT devices device and availability of edge/cloud servers. Therefore, making the optimal decision is the most critical issue for edge offloading. It needs to dynamically decide whether the task should be offloaded to the edge server or cloud server. If a large number of tasks are offloaded to the cloud server, the bandwidth will be occupied, which will greatly increase the transmission delay. Therefore, we need to have a reasonable offloading decision scheme so that it can reasonably allocate each task to the processing server. On the one hand, there are a large number of repetitive or similar tasks in the IoT environment, which often need to be retrained from scratch, resulting in inefficient offloading decision-making; on the other hand, some IoT dynamic scenarios have strict time constraints on task decision-making, and the slow learning speed of Convolutional Neural Network (CNN) is not suitable to meet the requirements of resource heterogeneity and real-time in the MEC system.

Faced with the rapidly changing IoT application scenarios, we cannot readjust the task offloading decision and wireless resource allocation through recalculation every time the MEC environment changes, otherwise, it will cause higher service delay and cost [4]. Although some good results have been achieved in offloading decision-making of MEC by introducing intelligent algorithms such as Reinforcement Learning (DRL) [5], there are still challenges such as the slow learning speed, and the failure of original network parameters when the model environment changes. In practical dynamic scenarios, the MEC environment is often affected by many factors anytime and anywhere. Conventional intelligent algorithms are usually based on neural networks. When the MEC environment changes, its original parameters will all fail and a large amount of training data is required to train from scratch, which makes the learning efficiency low. Such repeated training will consume resources and weaken the performance of the MEC system. At the same time, in order to improve efficiency, high

G. Qu and H. Wu are with the Center for Applied Mathematics, Tianjin University, Tianjin 300072, China (e-mail: {guanjinqu, whming}@tju.edu.cn).

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

P. Jiao is with the Center of Biosafety Research and Strategy, Law School, Tianjin University, Tianjin, 300350, China (e-mail: pjiao@tju.edu.cn).

(Corresponding author: Huaming Wu)

configuration equipment is also required to adapt to high-intensity training.

Considering the delay and energy consumption of IoT, offloading decisions can be made for a workflow with a series of dependent tasks. However, this kind of problem is generally NP-hard, traditional optimization methods are difficult to achieve results efficiently. One promising way of addressing the above issue is to bring deep learning techniques (especially DRL methods) into the computing paradigm of edge-cloud collaboration. Unfortunately, conventional DRL algorithms have the disadvantage of slower learning speed, which is mainly due to the weak inductive bias. A learning procedure with weak inductive bias will be able to adapt to a wide range of situations, however, it is generally less efficient [6].

To tackle the above challenges, we design an edge-cloud offloading framework in this paper, where IoT devices can choose to shift their computing tasks either to edge servers or cloud servers. Edge servers make offloading decisions based on task information for each device, reducing latency and energy consumption. We propose an efficient offloading decision-making method based on deep meta reinforcement learning [7] that takes advantage of DRL and meta-learning. To solve the problem of poor neural network portability, we introduce meta-learning to ensure that the offloading decision model can fast adapt to the new environment by learning the initial parameters of the neural network. The main contributions of this work are summarized as follows:

- We convert the dynamic computation offloading problem for dependent tasks under edge cloud computing into a multi-objective optimization problem. To jointly minimize the delay and energy consumption of IoT devices, we propose an effective and efficient offloading framework with intelligent decision-making capabilities.
- We design a novel Deep Meta Reinforcement learning-based Offloading (DMRO) framework that combines multiple parallel Deep Neural Networks (DNNs) and deep Q-learning algorithms to make offloading decisions. It includes an inner model and an outer model, where the former uses distributed DRL to find the optimal decision and the latter is trained with meta-learning to provide warm-start initialization for the inner model.
- Aiming at the change of MEC environments, an initial parameter training algorithm based on meta-learning is proposed, where meta-learning is applied to solve the problem of poor portability of DNNs. We conduct simulation experiments when considering different MEC task scenarios/environments, by learning the initial parameters of DNNs under various network environments, the offloading decision model achieves fast adaptation to a new MEC task environment.

The rest of the paper is organized as follows. In Section II, we review the related work. The system model and problem formulation are presented in Section III. The proposed Deep Meta Reinforcement learning-based Offloading (DMRO) framework is demonstrated in Section IV. Section V contains the simulation and its results. Finally, Section VI concludes the paper and draws future works.

II. Related work

MEC is an emerging computing paradigm, which can connect IoT devices to cloud computing centers through edge servers close to the device, thereby forming this task offloading mode in the IoT-edge-cloud computing environment [8]. By network functions virtualization and other means [9], the cloud center is responsible for providing flexible and on-demand computing resources for the execution of mobile applications, and the edge server is responsible for deciding which computing tasks need to be offloaded and providing a limited amount of computing resources. Thus, the energy consumption of the device and computing delay of the application can be reduced. In general, the task offloading process includes the following key components:

- *Application Partition*: Since different tasks usually have different amounts of computation and communication, before performing task offloading operation, it is better to divide the task into a workflow with multiple associated subtasks or as a series of independent subtasks, and then offload the subtasks separately. Among them, some subtasks are executed on the IoT devices, the others are executed on the relatively powerful server, making full use of the server resources, thereby greatly reducing the load of the IoT devices and improving their endurance [10].
- *Resource Allocation*: After the offloading decision is made, resources need to be allocated, including computing power, communication bandwidth, and energy consumption.

At present, task offloading algorithms related to decision-making can be divided into traditional methods and intelligent algorithms using artificial intelligence [4].

A. Traditional Offloading Decision-Making

Due to the NP-hardness of offloading decision problems in MEC, when the number of tasks increases, it is easy to encounter problems such as computational explosion. A diversity of platforms and frameworks like [11]–[14] have been proposed to solve the optimization problems of offloading binary decisions in edge-cloud environments.

A Lyapunov optimization framework was proposed in [15] to utilize open Jackson queuing network to formulate this joint optimization problem. eTime [16] was a cloud-to-device energy-efficient data transmission strategy based on Lyapunov-optimization, with more focus on data transmission optimization. Other studies using Lyapunov optimization for offloading decision-making can be found in [15], [17]–[19]. Markov processes and queueing models have been also widely applied for making offloading decisions. The offloading approach proposed in [20] supported two delayed offloading policies, i.e., a partial offloading model where jobs can leave the slow offloading to be executed locally, and a full offloading model where jobs can be offloaded directly via the cellular network. Besides, a computing offloading game theory has been developed in [21], which proposed a fast Stackelberg game

algorithm called C-SGA and a complex Stackelberg game algorithm called F-SGA to solve the decision problem of IoT-enabled cloud-edge computing. However, these optimization-based offloading algorithms can only obtain results after multiple iterations, which often involve too many complex calculation operations.

Conventional task offloading techniques usually apply some heuristic algorithms. A particle swarm optimization-based offloading decision algorithm was given in [22]. Xu *et al.* [23] proposed a computation offloading method called COM to solve the problem of computation offloading decisions in IoT-enabled cloud-edge computing environments. Goudarzi *et al.* [24] proposed a novel Memetic Algorithm (MA)-based application placement technique that can solve the task offloading problem in a multi-user multi-cloud multi-edge environment. However, heuristic algorithms are still difficult to solve complex problems that require a large amount of computation, and additional computation is also introduced, which results in high running time costs and energy consumption spent on offloading decision-making.

B. Intelligent Offloading Decision-Making

With the rapid development of computer science and the popularization of Artificial Intelligence (AI), deep learning has begun to be applied to solve the problem of offloading decision-making. Edge intelligence [2] or intelligent edge [25], that is, the convergence of edge computing and AI, takes advantage of both to achieve mutual benefit [26]. On the one side, optimizing DNNs through task offloading has become a new direction in edge intelligence research since edge computing can offload complex computing tasks to edge/cloud servers. On the other side, deep learning-driven approaches can facilitate offloading decision making, dynamic resource allocation and content caching, benefit in coping with the growth in volumes of communication and computation for emerging IoT applications [27].

Classic AI methods including deep learning and reinforcement learning, can provide more reasonable and intelligent solutions to solve the offloading decision problem in edge computing. Deep learning methods refer to the classification of the input task information through the multi-layer neural network to determine the final offloading position. Huang *et al.* [28] provided an algorithm that adopted distributed deep learning to solve the offloading problem of mobile edge networks. It used parallel and distributed DNNs to produce offloading decisions and achieved good results. A hybrid offloading model with the collaboration of Mobile Cloud Computing (MCC) and MEC was established in [29], where a distributed deep learning-driven task offloading (DDTO) algorithm was proposed to generate near-optimal offloading decisions over the IoT devices, edge cloud server, and central cloud server. Besides, Neurosurgeon [30] was a fine-grained partitioning method that can find the optimal dividing point in DNNs according to different factors, and made full use of the resources of cloud servers and mobile devices to minimize the computational delays or energy consumption in IoT environments.

In some cases, however, it is still difficult to treat task offloading decision-making as a classification problem to be solved by using deep learning techniques, which are mostly supervised learning. In addition, it is difficult to find labeled training sets for training on offloading decision problems. Reinforcement learning, as one of the paradigms of machine learning, is used to solve the interaction between the agent and the environment through learning, so as to achieve maximum return or specific goals. An edge-cloud task offloading framework using Deep Imitation Learning (DIL) [31] was proposed in [25], while training DNN model with DIL is still computation-intensive. Deep Reinforcement Learning (DRL) methods combined with neural network and reinforcement learning can be used to solve the task offloading decision problem in the MEC environment [32]. The final decision is the maximum reward action under the interaction with the environment. The premise of using DRL algorithms for offloading decision-making is that it can be regarded as a Markov process, in which three spaces named state, action, and reward are established. Among them, the task information is input into the state, and the offloading decision is located in the action space. Zhang *et al.* [33] proposed an offloading decision scheme based on the Actor-Critic algorithm. In [34] and [5], task offloading decisions were made based on DRL algorithms, e.g., Deep Q-Learning Network (DQN) and Double Deep Q-learning Network (DDQN)-based algorithms, however, the cloud server was not considered in the MEC environment and they usually require to learn from scratch when the environment changes.

Currently, the role of DRL is to choose an optimal edge computing environment or location for the current task according to its status and environment. However, each time the IoT environment changes, the offloading decision has to be recalculated, which leads to more service delays and higher costs. In addition, DRL algorithms are still limited with slower learning speed and are generally less efficient in solving the offloading decision-making problem [6]. Table I shows the comparison of the key parameters of the current relevant unloading decision model. It can be seen that although it uses different algorithms, it does not consider the adaptability of the model when the scenario/environment varies. Therefore, it is urgent to find an intelligent method that can learn knowledge and quickly provide better offloading decisions with the change of environment. Unlike traditional machine learning that only trains a general learning model for edge offloading, the goal of meta-learning is "learning to learn fast", that is, to make the model become a learner and fit a new environment rapidly [35]–[37]. After completing multiple learning tasks, it can quickly learn new tasks by learning prior knowledge or exploring learning strategies only with a few training examples [38], [39]. Therefore, it adapts to complex and dynamic environments rapidly and can be used to improve the robustness of task offloading decisions in IoT environments.

III. System Model and Problem Formulation

In this section, we give an overview of the system model and then define the delay model and energy consumption

TABLE I: The Qualitative Comparison of the Current Literature

Techniques	IoT Properties		Architectural Properties			Model Properties				
	Dependency Mode	IoT Number	Edge Number	Cloud Number	Network structure	Algorithm Properties	Theories	Decision Parameters		Fast Adaptability
								Time	Energy	
[11]	Dependent	Multiple	Multiple	Multiple	MEC+MCC	Traditional	Coalition Game theory	✗	✓	✗
[15]	Dependent	Multiple	Multiple	Multiple	MEC+MCC	Traditional	Lyapunov Optimization	✓	✓	✗
[16]	Independent	Multiple	-	Single	MCC	Traditional	Lyapunov Optimization	✓	✓	✗
[20]	Independent	Multiple	-	Multiple	MCC	Traditional	Markov Chain	✓	✓	✗
[21]	Independent	Multiple	Multiple	-	MEC	Traditional	Stackelberg Game	✓	✗	✗
[24]	Dependent	Multiple	Multiple	Multiple	MEC+MCC	Traditional	Genetic Algorithm	✓	✓	✗
[28]	Independent	Multiple	Single	-	MEC	Intelligent	Deep Learning	✓	✓	✗
[29]	Independent	Multiple	Single	Single	MEC+MCC	Intelligent	Deep Learning	✓	✓	✗
[33]	Independent	Single	Multiple	-	MEC	Intelligent	Deep Reinforcement Learning	✓	✓	✗
[5]	Independent	Multiple	Single	-	MEC	Intelligent	Deep Reinforcement Learning	✓	✗	✗
[34]	Independent	Multiple	Single	Single	MEC+MCC	Intelligent	Deep Reinforcement Learning	✓	✓	✗
Our Technique	Dependent	Multiple	Single/Multiple	Single/Multiple	MEC+MCC	Intelligent	Deep Meta Reinforcement Learning	✓	✓	✓

model. On this basis, the optimization problem of computation offloading is formulated.

A. System Model

The system model for task offloading in IoT-edge-cloud computing environments is shown in Fig. 1. The proposed framework is composed of a cloud server, an edge server, and multiple IoT devices, where the IoT devices can either execute locally or offload their workflow to the cloud server or edge server.

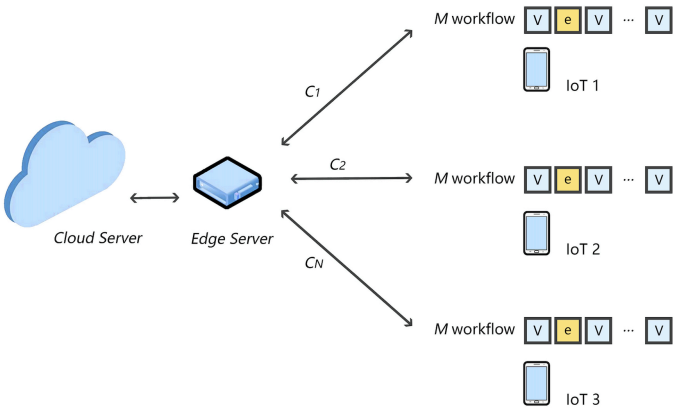


Fig. 1: System model of edge-cloud computing with multiple IoT devices

In this framework, edge servers are distributed near the devices and have high bandwidth. The edge server accepts workflow information from the device and makes fine-grained offloading decisions. The program for each device can be divided into sequential workflows. We assume the x-th workflow is defined as follows:

$$R_x = \{e_{0,1}, v_1, e_{1,2}, v_2, \cdots, v_i, e_{i,j}, v_j, \cdots, e_{n-1,n}, v_n, e_{n+1}\}, \tag{1}$$

where v_i denotes i-th task in the workflow, and $e_{i,j}$ illustrates the set of data flows between tasks v_i and v_j.

Each workflow x can determine whether to offload its task v_i or not, and the offloading decision is denoted by a Matrix variable:

$$b_{x,i} \in (b_0, \quad b_1, \quad b_2), \tag{2}$$

where $b_0 = [1 \quad 0 \quad 0]^T$, $b_1 = [0 \quad 1 \quad 0]^T$ and $b_2 = [0 \quad 0 \quad 1]^T$ denote the decision that workflow x to execute its i-th task locally, offload i-th task to the edge server, and offload i-th task to the cloud server, respectively.

B. Delay Model

The delay caused by computation offloading includes computation delay and transmission delay. We do not consider the delay incurred in offloading decision-making because the time required to make the decision is short. Therefore, the computational delay of task v_i is calculated by:

$$T_i^c = \begin{cases} \frac{v_i}{C_0}, & b_{x,i} = b_0, \\ \frac{v_i}{C_1}, & b_{x,i} = b_1, \\ \frac{v_i}{C_2}, & b_{x,i} = b_2, \end{cases} \tag{3}$$

where C_0, C_1 and C_2 stand for the computing power of the IoT, the computing power of the edge server and the computing power of the cloud server, respectively.

The transmission delay between tasks v_i and v_j is:

$$T_{i,j}^t = \begin{cases} 0, & b_{x,i} = b_{x,j}, \\ \frac{e_{i,j}}{B_{0,1}}, & b_{x,i} = b_0, b_{x,j} = b_1 \text{ or } b_{x,i} = b_1, b_{x,j} = b_0, \\ \frac{e_{i,j}}{B_{1,2}}, & b_{x,i} = b_1, b_{x,j} = b_2 \text{ or } b_{x,i} = b_2, b_{x,j} = b_1, \\ \frac{e_{i,j}}{B_{0,2}}, & b_{x,i} = b_0, b_{x,j} = b_2 \text{ or } b_{x,i} = b_2, b_{x,j} = b_0, \end{cases} \tag{4}$$

where $B_{0,1}$ denotes the allocated bandwidth between the IoT device and the edge server. $B_{1,2}$ is the allocated bandwidth between the cloud server and the edge server. Similarly, we denote $B_{0,2}$ as the allocated bandwidth between the IoT device and the cloud server.

The total delay for workflow x is calculated as:

$$T_x = \sum_{i=1}^{N} \left(T_i^c + T_{i,i+1}^t\right), \tag{5}$$

where the workflow x has N associated tasks.

C. Energy Consumption Model

The energy consumption model of workflow x can be expressed as:

$$E_x = E_x^{\text{local}} + \alpha E_x^{\text{edge}} + \beta E_x^{\text{cloud}}, \tag{6}$$

where α and β are weights of the energy consumption at the edge server and at the cloud server, respectively. When $\alpha = \beta = 0$, we only consider the energy consumption at the IoT device. For simplicity, we ignore the energy consumed during task transmission.

The energy consumption of task v is calculated as:

$$E_i = \begin{cases} v_i \cdot d_{\text{local}}, & b_{x,i} = b_0, \\ v_i \cdot d_{\text{edge}}, & b_{x,i} = b_1, \\ v_i \cdot d_{\text{cloud}}, & b_{x,i} = b_2, \end{cases} \tag{7}$$

where d_{local}, d_{edge} and d_{cloud} denote the local energy consumption per data bit, the edge energy consumption per data bit and the cloud energy consumption per data bit, respectively.

Therefore, the energy consumption model of workflow x can be expressed by:

$$E_x = \sum_{i=1}^{N} [E_i, \alpha E_i, \beta E_i] \cdot b_{x,i}. \tag{8}$$

D. Problem Formulation

To minimize both the delay for completing all workflows and the corresponding energy consumption simultaneously, we first introduce a system utility $Q(x, b)$, which is defined as the weighted sum of energy consumption and workflow completion delay, as follows:

$$\begin{aligned} Q(x,b) &= \sum_{x=1}^{M} (T_x + \delta E_x) \\ &= \sum_{x=1}^{M} \left[\sum_{i=1}^{N} \left(T_i^c + T_{i,i+1}^t\right) + \delta \sum_{i=1}^{N} [E_i, \alpha E_i, \beta E_i]\, b_{x,i} \right], \end{aligned} \tag{9}$$

where there are M workflows in total, each workflow has N associated tasks, and δ denotes the weight of energy consumption and task completion time.

The optimization problem can be formulated as a minimization problem $\mathcal{P}_1$ with a constraint, as follows:

$$(\mathcal{P}_1): \quad \min_{b} \; Q(x,b), \tag{10}$$

$$\text{s.t.}: \quad b_{x,i} \in \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \right). \tag{11}$$

IV. Deep Meta Reinforcement Learning-based Offloading Framework

To effectively solve the optimization problem defined in (10), we then propose a Deep Meta Reinforcement learning-based Offloading (DMRO) framework as shown in Fig. 2, where a series of dependent tasks are considered comprehensively, in order to give a specific offloading decision for each task. The proposed learning-driven offloading framework contains a task offloading decision model based on distributed reinforcement learning algorithm and a training model based on meta-learning, aiming to solve the problem of poor portability of neural networks.

The DMRO framework can be divided into two types of models, i.e., an inner model and an outer model. The purpose

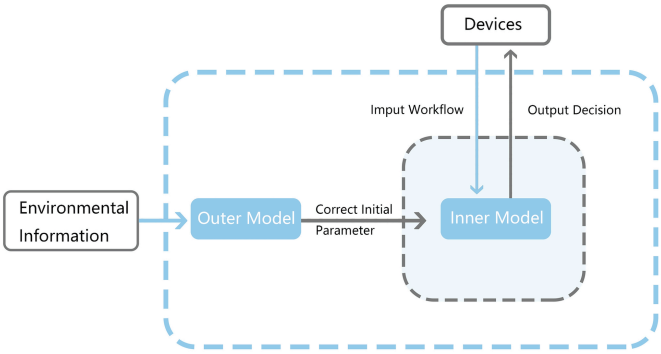


Fig. 2: The proposed deep meta reinforcement learning-based offloading framework

of the framework using inner and outer models is to improve the interaction between the model and the environment on the traditional task offloading decision model. The inner model focuses on making unloading decisions. The outer model focuses on unloading changes in the environment and changes the model parameters of the inner layer according to the changes, so that the inner model can be adjusted more quickly. Specifically, the inner model is an offloading decision model, which is responsible for receiving the workflow and training the model parameters to provide final offloading decisions for different tasks. The outer model is responsible for training the initial parameters to improve the portability of the model. When the environment of the MEC system changes, e.g., the performance of the edge server or the bandwidth between the IoT device and the edge server, it can adjust the parameters of the neural network in the inner model, so that the system can quickly adapt to the new environment. When the workflow is input into the edge offloading system, the outer model first determines whether the external environment has changed, in order to determine whether to adjust the initial parameters. After that, the workflow will enter the inner model, which will make the offloading decision, and store the state and action in memory for the training of the neural network.

Furthermore, in order to increase the portability of the model, speed up the decision-making process and reduce the amount of computation, we design a deep meta-reinforcement learning-based method, which also combines the function of memory playback (replay memory), so that the decision-making system can adapt to the new environment quickly and give the optimal offloading decision when the environment changes. In addition, the generated offloading decisions are stored in the replay memory summary for further learning.

A. Inner Model

As shown in Fig. 3, the inner model is based on a parallel Deep Reinforcement Learning (DRL) algorithm. We apply a classic reinforcement learning method named Q-learning, in which we input environmental parameters, labeled initial parameters and workflow x into the inner model.

We use a_i to represent the offloading decision of the i-th

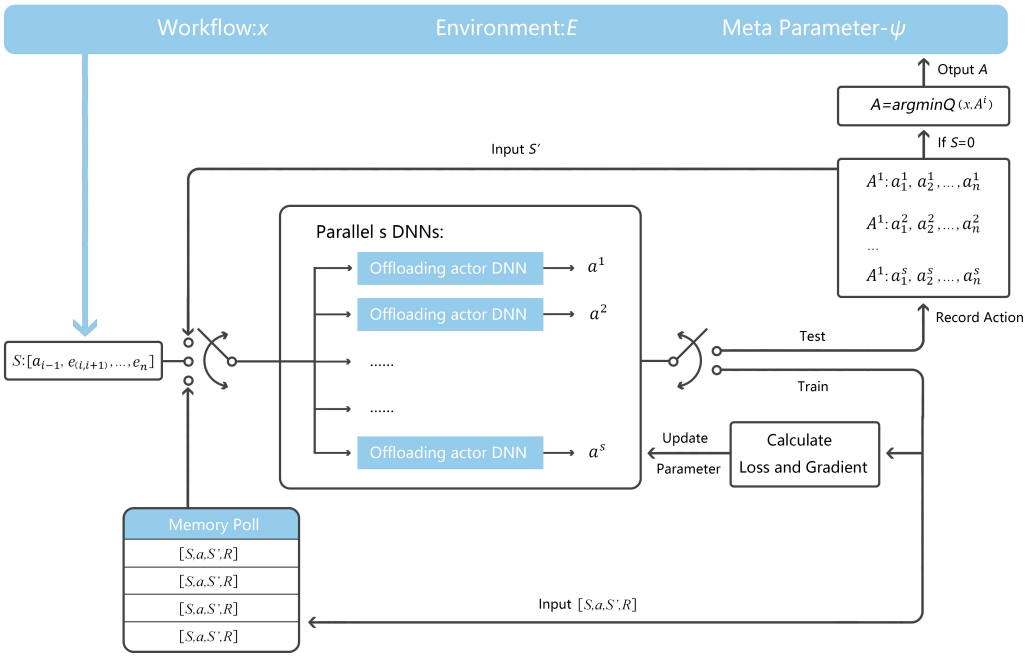


Fig. 3: Illustration of the proposed distributed DRL-based offloading scheme

subtask of the workflow, which is defined as:

$$a_i = \begin{cases} 0, & \text{if subtask } i \text{ is executed on IoT device,} \\ 1, & \text{if subtask } i \text{ is offloaded to edge server,} \\ 2, & \text{if subtask } i \text{ is offloaded to cloud server,} \end{cases} \tag{12}$$

where $a_i = 0$, 1, and 2 indicate that the i-th subtask is executed locally on the IoT device, the edge server, and the cloud server, respectively.

We represent S_i as the state when processing the i-th subtask in the workflow:

$$S_i = [a_{i-1}, e_{i-1,i}, v_i, e_{i,i+1}, v_{i+1}, \cdots, e_{n-1,n}, v_n, e_n], \ i \geq 1, \tag{13}$$

where a_{i-1} represents the execution position of a subtask in the workflow, which is set as 0 at the beginning. Then the state S_i is input to the neural network to find the Q value of each action in this state.

Here we have s distributed neural network units. Each neural network action unit is parallel, including two DNNs with an identical structure, one of which is the target network for parameter freezing. Parameter freezing means that the two networks have the same structure, but the parameters of the frozen network will not be iterated every time. When the other network learns a certain number of times, the parameters are copied to the frozen network. The purpose of using parameter freezing is to reduce the relevance of learning [40]. Each neural network unit will give the selected action value according to its own Q value calculated by the greedy algorithm. In addition, we define a local objective function:

$$F(S_i, a) = T_i^c + T_{i-1,i}^t + \delta E_i, \tag{14}$$

$$b_{x,i-1} = a_{i-1}, \tag{15}$$

$$b_{x,i} = a_i, \tag{16}$$

where $F(S_i, a)$ can be interpreted as the weighted sum of the delay and energy consumption for selecting action a in state S_i. We compare $F(S_i, a)$ values generated by the actions selected by different DNNs as a measure of the effects of the actions selected by different DNNs. The action with the lowest F is set as the optimal solution in the state S_i.

For the reward function $R(S_i, a)$ in DRL, if the action is the action value of the optimal solution, the reward value is the negative value of the minimum optimization function; otherwise, the reward value is the negative value of the maximum function. Then we choose the action of the optimal solution as a_i, and update the state S_i as:

$$S_{i+1} = [a_i, e_{i,i+1}, v_{i+1}, e_{i+1,i+2}, v_{i+2}, \cdots, e_{n-1,n}, v_n, e_n]. \tag{17}$$

The algorithmic process of the proposed parallel DRL algorithm is as demonstrated in **Algorithm** 1.

1) Training Phase: In the training phase, we input $[S_i, a_i, R(S_i, a), S_{i+1}]$ calculated by the neural network into the memory, and then continue to input the updated workflow into the neural network for calculation until all subtasks of the workflow have been processed.

After processing a certain number of workflows, e.g., five times, we will randomly extract $[S_i, a_i, R(S_i, a), S_{i+1}]$ from the memory for empirical playback. The purpose of this step is to eliminate the correlation generated by the associated states. Then we update the parameters of the network as follows:

$$Q(S_i, a_i) \leftarrow (1-\theta) Q(S_i, a_i) + \theta \left[R(S_i, a) + \mu \max_{a' \in A} Q(S_{i+1}, a') \right] \tag{18}$$

where $Q(S_i, a_i)$ represents the Q value function, which is calculated by the neural network, $Q(S_i, a_i)$ represents the Q part is calculated by the network with the latest parameters,

and $\max_{a' \in A} Q(S_{i+1}, a')$ is calculated by the network with the frozen parameters. The learning rate $\theta \in [0, 1]$ is the weight of the current offloading experience. The discount factor $\mu \in [0, 1]$ denotes the short view of the IoT device regarding the future reward.

2) Decision-Making Phase: In the decision-making phase, we will make fine-grained offloading decisions for IoT devices. First, we obtain the action value a generated by each DNN and fill it into s action sets A. Then, we input the updated state to the neural network, and continue to find the execution method of the next subtask until all the subtasks are assigned. At this time, A^i represents the offloading scheme given by the i-th DNN network to the workflow x, and the scheme A^i with the minimum $Q(x, b)$ value is the final scheme A and output to the device.

Algorithm 1 The distributed deep reinforcement learning algorithm

Input: Workflow x, Environment: E, Meta-parameter: ψ
Output: Optimal offloading decision A

Initialize the s DNNs with meta-parameter ψ
Empty the memory pool
for $i = 1, 2, 3, \cdots, N$ **do**
  Replicate state S_i to all s DNNs
  Generate s-th offloading action a_i^j via ϵ-greedy policy
  for $j = 1, 2, 3, \cdots, s$ **do**
    Input a_i^j to decision set A^j as: a_i^1, a_i^2, $\cdots$, a_i^j
    Evaluate the local objective function $F(S_i, a_i^j)$, generate reward $R(S_i, a_i^j)$
    if train **then**
      $a_i^1 = a_i$
    else
      a_i is a_i^j in turn
      if $i == n$ **then**
        Select A according to $\arg\min_{A^j} Q(S_i, A^j)$
        Output A as offloading decision
      end if
      Input $[S_i, a_i, R(S_i, a_i), S_{i+1}]$ to memory pool
    end if
  end for
  if Add data to memory five times **then**
    Extract $[S_i, a_i, R(S_i, a_i), S_{i+1}]$ from memory at random
    Replicate state S_i to all s DNNs
    Evaluate the local objective function $F(S_i, a_i)$, generate reward $R(S_i, a_i)$
    Update the s DNNs weights θ
  end if
end for

B. Outer Model

In the outer model, we propose a meta algorithm to learn the initial parameters.

Based on the original algorithm described in [38], i.e., an initial parameter algorithm for training different image classification networks, we propose a novel algorithm for learning initial parameters in order to adapt to the training method of reinforcement learning. We train our decision-making engine by leveraging the deep meta-learning method, and then we make rapid offloading decisions through IoT-edge-cloud computing environments. The algorithmic process of the proposed meta algorithm is as listed in **Algorithm** 2.

The principle of our meta algorithm is to input the decision-making and execution results of the workflow in different environments into the training model. Each time the training model randomly selects training samples in one environment for learning, and randomly selects another environment after learning. The purpose of training sample learning is to ensure that the parameters trained by the model will not be too close to the optimal solution in a specific environment. We use the parameters trained in this way as the initial parameters of the inner model.

Algorithm 2 The meta algorithm

Input: Workflow x, Environment: E
Output: Optimal offloading decision A

Initialize the DNNs with parameter θ_0
Empty the memory pool
for $i = 1, 2, 3, \cdots, N$ **do**
  Randomly select environment
  Input state S_i to DNN
  Generate offloading action a_i via ϵ-greedy policy
  generate reward $R(S_i, a_i)$ via Random Environment
  Input $[S_i, a_i, R(S_i, a_i), S_{i+1}]$ to memory pool
  if Add data to memory five times **then**
    Randomly select environment
    Extract $[S_i, a_i, R(S_i, a_i), S_{i+1}]$ from memory at random
    Replicate state S_i to the DNNs
    Evaluate the local objective function $F(S_i, a_i)$, generate reward $R(S_i, a_i)$
    Update the s DNNs weights θ
  end if
end for
Output DNN parameter θ as meta-parameter ψ

V. Performance Evaluation

In this section, we evaluate the performance of the proposed DMRO framework for solving the optimization problem of offloading decision-making under different MEC environments.

A. Experimental Setup

In our simulation, we assume that there are four IoT users, and each user has five workflows. The size of the first subtask of each workflow is $50 - 100$ MB, and the size of subsequent tasks is $10 - 50$ MB. The amount of computation for each subtask is $10^3 - 10^5$ MHz randomly distributed. For the DNN structure, we consider a fully connected DNN consisting of one input layer, two hidden layers, and one output layer in the proposed DMRO framework. Since this paper is not limited to a special environment sensitive to power consumption, the parameters α and β are both set to 1. In addition, we set the environmental information as listed in Table II.

TABLE II: Environmental Information

C_{local}	30 MHz	$B_{0,1}$	800 MB/s	d_{local}	0.3 J/MB
C_{edge}	70 MHz	$B_{1,2}$	200 MB/s	d_{edge}	0.15 J/MB
C_{cloud}	150 MHz	$B_{0,2}$	10 MB/s	d_{cloud}	0.1 J/MB

B. Convergence Performance

Figure 4 shows the convergence performance of our model under different learning rates, where the abscissa is the number of training steps and the ordinate is the value of the neural network loss function. Due to the training methods of reinforcement learning, this is not a true loss function, so we use "cost" instead. It can be found that when the learning rate is too low, it will not be able to converge. However, when the learning rate is 0.01, the convergence effect is the best, so we will use a learning rate of 0.01 in the next experiment.

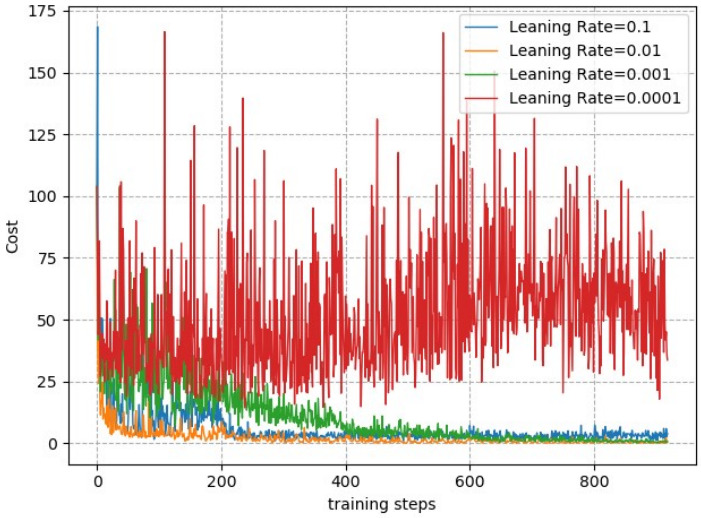


Fig. 4: Convergence performance under different learning rates

Figure 5 shows the convergence performance of our model under different batch sizes, where the abscissa is the number of training steps and the ordinate is the value of the neural network loss function. It can be seen that the batch size has less effect on the convergence, but as the batch size increases, the volatility of the curve becomes smaller. It is worth noting that there is a small fluctuation in the curve every 200 steps, which is mainly due to the parameter freezing mechanism. In the model, we set the network parameters to the target network every 200 steps. As a result, the parameters will fluctuate every 200 steps, but it does not affect the convergence of the model.

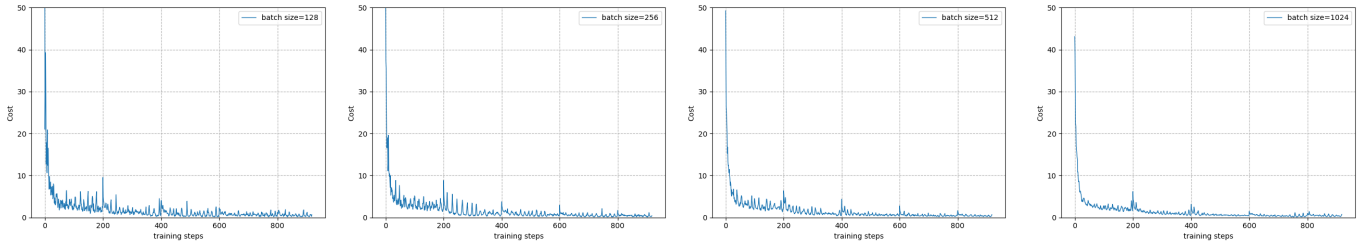
C. Comparison with Related Work

To gain insight into the proposed DMRO scheme for edge offloading decision, we have implemented many existing approaches in heterogeneous IoT-edge-cloud computing environments for comparison:

- *Local-only scheme*: In this method, all tasks of workflows are executed locally on the IoT device. The results of this method can be used as a benchmark to analyze the gain of task offloading techniques.
- *Edge-only scheme*: This is a full offloading scheme. In this method, all tasks of workflows are fully offloaded to the edge servers for further processing.
- *Cloud-only scheme*: This is a full offloading scheme. In this method, all tasks of workflows are fully offloaded to the cloud server for further processing.
- *Deep Q-Network scheme*: This is a partial offloading scheme based on the Deep Q-Network (DQN) algorithm [41], where it can be regarded as a simplified DMRO algorithm with only one parallel network. As one of the most classic algorithms in deep reinforcement learning, DQN is frequently used in task offloading [42]. In this method, we use the Deep Q-Network in making dynamic offloading decisions.
- *Double DQN scheme*: Double DQN is an improved algorithm based on nature Deep Q-Network [43]. It uses two identical neural networks to solve the correlation between data samples and network before training. In [44], the task offloading model uses Double DQN to improve the processing capacity of the edge node and reduce the packet loss rate and average delay.
- *Dueling DQN scheme*: Dueling DQN is another improved algorithm based on nature Deep Q-Network [45], which is achieved by optimizing the experience playback pool and sampling by weight. There is also some work that introduces Dueling DQN into task offloading [46].
- *DMRO scheme*: This is a partial offloading scheme based on the proposed DMRO scheme. It is designed to find the optimal offloading scheme that minimizes the weighted delay and energy consumption.

In order to control the variables, several other deep reinforcement learning offloading schemes used in the experiment are the same as the initial parameters of this scheme, including the same state space and action space as well as the same neural network structure. The comparison results are as shown in Fig. 6, where the abscissa is the weight ratio of delay to energy consumption, and the ordinate is the value of the objective function. Since we used feature scaling, the value of the objective function is dimensionless. The lower the total cost, the smaller the resulting delay and energy consumption, which indicates a better offloading effect. Especially, when the weight value is 0, it means that only delay is considered. The figure shows that the DMRO algorithm can achieve the minimum total cost among the seven methods. The offloading algorithm based on reinforcement learning is better than the single offloading method. DMRO shows a similar curve trend with DQN, and outperforms the *Deep Q-Network scheme*, the *Double DQN scheme*, and the *Dueling DQN scheme*. As can be seen from the figure, the DMRO algorithm is generally better than other algorithms. For example, when the weight is 0.2, the DMRO algorithm improves the effect by more than 17.6% compared with other DRL algorithms. In addition, as the weight ratio of energy consumption increases, the total consumption of local execution increases rapidly, which also meets our expectations, indicating that local devices are more sensitive to energy consumption.

Apart from the scenario with one edge server and one



(a) Batch size 128 (b) Batch size 256 (c) Batch size 512 (d) Batch size 1024

Fig. 5: Convergence performance under different batch sizes

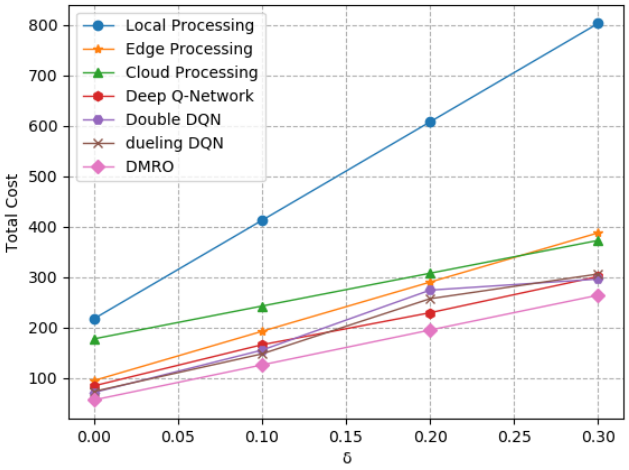


Fig. 6: Performance comparison of different offloading schemes under different weights

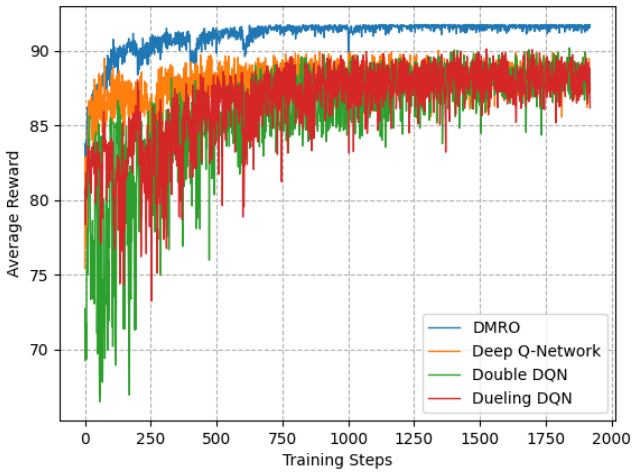


Fig. 7: Performance comparison of different offloading schemes in a multi-server environment

cloud server, the DMRO framework is highly scalable and can be easily extended to other dynamic scenarios with multiple edge servers or multiple cloud servers. Here, to examine the effect of DMRO in a multi-cloud server and multi-edge server environment, we simulate an environment with two edge servers, and two cloud servers. Table III gives the detailed parameter settings, where different servers have different

TABLE III: Multi-Server Environmental Information

C_{local}	30 MHz	$B_{0,1}$	800 MB/s	d_{local}	0.3 J/MB
C_{edge_1}	70 MHz	$B_{1,2}$	200 MB/s	d_{edge_1}	0.15 J/MB
C_{edge_2}	90 MHz	$B_{1,1}$	500 MB/s	d_{edge_2}	0.18 J/MB
C_{cloud_1}	150 MHz	$B_{0,2}$	10 MB/s	d_{cloud_1}	0.1 J/MB
C_{cloud_2}	180 MHz	$B_{2,2}$	150 MB/s	d_{cloud_2}	0.08 J/MB

computing power and bandwidth. In this environment, we still consider the offloaded energy consumption and latency as the objective function. Figure 7 compares the effects of different DRL algorithms in the multi-edge environment, where the horizontal coordinate is the number of training steps and the vertical coordinate is the average reward value in the DRL algorithm. Since the reward value of this model is the negative correlation of the objective function value, a higher average reward represents a higher model offloading effect. From the figure, we can see that all DRL algorithms improve with the increase of training steps, which proves that we have set a reasonable state space and reward function. Moreover, the proposed DMRO scheme not only converges quickly, but also achieves the highest average reward value, which demonstrates that the offloading results using DMRO are significantly better than other DRL algorithms such as *Deep Q-Network scheme*, *Double DQN scheme* and *Dueling DQN scheme*.

D. Fast Learning

We show the effect of the proposed meta algorithm in fast offloading decision learning under different MEC environments. We first set up two types of environments, i.e., the training environment and the testing environment, as shown in Table IV.

TABLE IV: Evaluation Parameters

Training Environment		Testing Environment	
C_{local}	$15-25$ MHz	C_{local}	30 MHz
C_{edge}	$50-60$ MHz	C_{edge}	70 MHz
C_{cloud}	$160-170$ MHz	C_{cloud}	150 MHz
$B_{0,1}$	$600-700$ MB/s	$B_{0,1}$	800 MB/s
$B_{1,2}$	$100-150$ MB/s	$B_{1,2}$	200 MB/s
$B_{0,2}$	$20-30$ MB/s	$B_{0,2}$	10 MB/s
d_{local}	0.3 J/MB	d_{local}	0.3 J/MB
d_{edge}	0.15 J/MB	d_{edge}	0.15 J/MB
d_{cloud}	0.1 J/MB	d_{cloud}	0.1 J/MB

We input the trained meta parameters into the test environment. As depicted in Fig. 8, we compare the convergence of the meta-parameters, the initialization parameters, and the parameters that have been trained in other similar environments. The abscissa is the number of training steps, and the ordinate is the value of the neural network loss function. It can be seen that the convergence of the meta-parameters is significantly better than that of the traditional initialization parameters and similar to that of the trained parameters. This indicates that both the meta-learning and the trained parameters have strong convergence in the new MEC environment.

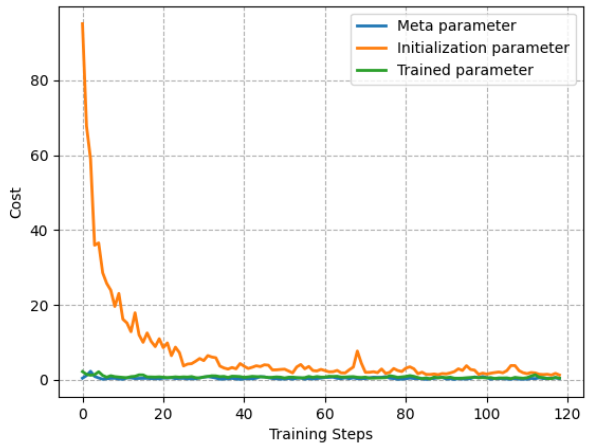


Fig. 8: Convergence performance under different parameters.

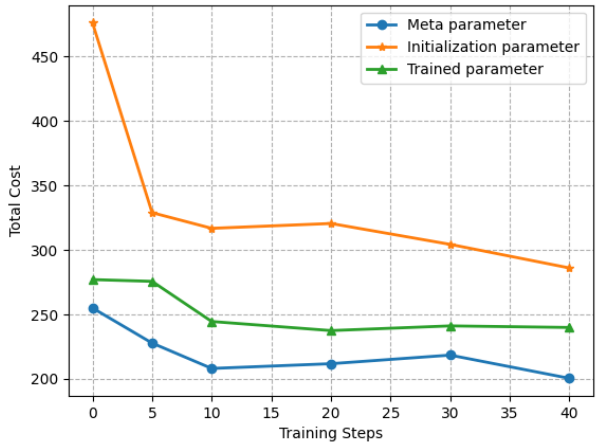


Fig. 9: Performance comparison under different parameters.

Figure 9 shows the comparison of the effect of the meta parameters, initialization parameters and trained parameters on the total cost. The abscissa is the weight ratio of delay to energy consumption, and the ordinate is the value of the objective function. It can be seen from the figure that the decision result of DNNs using the meta parameters is significantly better than that of the traditional initialization parameters as well as the trained parameters. After several rounds of training, the optimal offloading decision can be achieved. Therefore, when the environment of the model changes, although both the meta parameters and the trained parameters are close to convergence, using the meta parameters can obtain better results than the original trained parameters.

VI. Conclusion and Future Work

To solve the challenge of poor portability of neural networks, this paper has proposed a novel DMRO framework to deal with the task offloading decision-making problem in heterogeneous edge and cloud collaborative computing environments. It includes a task offloading decision model based on distributed DRL and a training initial parameter model based on deep meta-learning, which has the potential to fast adapt to a dynamic MEC environment and solve the problem of offloading decision-making for edge-cloud computing.

Experimental results show that DMRO has a better effect on task offloading decisions than binary offloading schemes and conventional DRL-based partial offloading schemes. In addition, due to the use of meta parameters, the model has stronger portability and rapid environment learning ability. Once the MEC environment changes, the model can quickly converge, and only a small number of learning steps are required to find optimal offloading solutions with relatively low costs.

In future work, we will continually improve the meta-learning algorithm so that it can better adapt to task offloading decisions in large-scale MEC environments, especially, the initial parameters can be changed adaptively in response to environmental parameters. In addition, we will also focus on issues such as resource allocation and bandwidth adjustment through serverless edge computing frameworks [47], [48]. Furthermore, the offloading model can not only give the offloading decision result of tasks, but also provide the corresponding resource scheduling schemes.

Acknowledgment

This work was partly supported by the National Natural Science Foundation of China under Grant No. 62071327 and 61801325, Natural Science Foundation of Tianjin City under Grant No. 18JCQNJC00600 and JSPS KAKENHI under Grant No. JP19H04105.

References

[1] Y. Ai, M. Peng, and K. Zhang, "Edge computing technologies for internet of things: a primer," *Digital Communications and Networks*, vol. 4, no. 2, pp. 77–86, 2018.

[2] D. Xu, T. Li, Y. Li, X. Su, S. Tarkoma, and P. Hui, "A survey on edge intelligence," *ArXiv*, vol. abs/2003.12172, 2020.

[3] H. Wu, Y. Sun, and K. Wolter, "Energy-efficient decision making for mobile cloud offloading," *IEEE Transactions on Cloud Computing*, vol. 8, no. 2, pp. 570–584, Apr. 2020.

[4] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[5] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, 2019.

[6] M. Botvinick, S. Ritter, J. X. Wang, Z. Kurth-Nelson, C. Blundell, and D. Hassabis, "Reinforcement learning, fast and slow," *Trends in cognitive sciences*, vol. 23, no. 5, pp. 408–422, 2019.

[7] J. X. Wang, Z. Kurth-Nelson, D. Tirumala, H. Soyer, J. Z. Leibo, R. Munos, C. Blundell, D. Kumaran, and M. Botvinick, "Learning to reinforcement learn," 2017.

[8] I. A. Elgendy, W.-Z. Zhang, Y. Zeng, H. He, Y.-C. Tian, and Y. Yang, "Efficient and secure multi-user multi-task computation offloading for mobile-edge computing in mobile IoT networks," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2410–2422, Dec. 2020.

[9] P. P. Ray and N. Kumar, "SDN/NFV architectures for edge-cloud oriented IoT: A systematic review," *Computer Communications*, vol. 169, pp. 129–153, mar 2021.

[10] H. Wu, W. Knottenbelt, and K. Wolter, "An efficient application partitioning algorithm in mobile environments," *IEEE Transactions on Parallel and Distributed Systems*, vol. 30, no. 7, pp. 1464–1480, 2019.

[11] N. Kumar, T. Dhand, A. Jindal, G. S. Aujla, H. Cao, and L. Yang, "An edge-fog computing framework for cloud of things in vehicle to grid environment," in *2020 IEEE 21st International Symposium on "A World of Wireless, Mobile and Multimedia Networks" (WoWMoM)*, 2020, pp. 354–359.

[12] C. Liu, K. Li, J. Liang, and K. Li, "COOPER-SCHED: A cooperative scheduling framework for mobile edge computing with expected deadline guarantee," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–1, 2020.

[13] A. Irshad, S. A. Chaudhry, O. A. Alomari, K. Yahya, and N. Kumar, "A novel pairing-free lightweight authentication protocol for mobile cloud computing framework," *IEEE Systems Journal*, pp. 1–9, 2020.

[14] C. Liu, K. Li, J. Liang, and K. Li, "COOPER-MATCH: Job offloading with a cooperative game for guaranteeing strict deadlines in MEC," *IEEE Transactions on Mobile Computing*, pp. 1–1, 2020.

[15] K. Peng, J. Nie, N. Kumar, C. Cai, J. Kang, Z. Xiong, and Y. Zhang, "Joint optimization of service chain caching and task offloading in mobile edge computing," *Applied Soft Computing*, vol. 103, p. 107142, May 2021.

[16] P. Shu, F. Liu, H. Jin, M. Chen, F. Wen, Y. Qu, and B. Li, "eTime: Energy-efficient transmission between cloud and mobile devices," in *2013 Proceedings IEEE INFOCOM*. IEEE, 2013, pp. 195–199.

[17] 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, 2016.

[18] G. Zhang, W. Zhang, Y. Cao, D. Li, and L. Wang, "Energy-delay tradeoff for dynamic offloading in mobile-edge computing system with energy harvesting devices," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 10, pp. 4642–4655, 2018.

[19] 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," *Internet of Things Journal*, vol. 8, no. 4, pp. 2163–2176, feb 2021.

[20] H. Wu and K. Wolter, "Stochastic analysis of delayed mobile offloading in heterogeneous networks," *IEEE Transactions on Mobile Computing*, vol. 17, no. 2, pp. 461–474, 2018.

[21] M. Li, Q. Wu, J. Zhu, R. Zheng, and M. Zhang, "A computing offloading game for mobile devices and edge cloud servers," *Wireless Communications and Mobile Computing*, vol. 2018, 2018.

[22] J. Bi, H. Yuan, S. Duanmu, M. Zhou, and A. Abusorrah, "Energy-optimized partial computation offloading in mobile-edge computing with genetic simulated-annealing-based particle swarm optimization," *IEEE Internet of Things Journal*, vol. 8, no. 5, pp. 3774–3785, mar 2021.

[23] X. Xu, Q. Liu, Y. Luo, K. Peng, X. Zhang, S. Meng, and L. Qi, "A computation offloading method over big data for IoT-enabled cloud-edge computing," *Future Generation Computer Systems*, vol. 95, pp. 522–533, 2019.

[24] M. Goudarzi, H. Wu, M. Palaniswami, and R. Buyya, "An application placement technique for concurrent iot applications in edge and fog computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 4, pp. 1298–1311, 2021.

[25] 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 Communications*, vol. 27, no. 1, pp. 92–99, Feb. 2020.

[26] 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 Communications Surveys Tutorials*, vol. 22, no. 2, pp. 869–904, 2020.

[27] B. Cao, L. Zhang, Y. Li, D. Feng, and W. Cao, "Intelligent offloading in multi-access edge computing: A state-of-the-art review and framework," *IEEE Communications Magazine*, vol. 57, no. 3, pp. 56–62, 2019.

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

[29] 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 of Things Journal*, vol. 7, no. 9, pp. 8099–8110, 2020.

[30] Y. Kang, J. Hauswald, C. Gao, A. Rovinski, T. Mudge, J. Mars, and L. Tang, "Neurosurgeon: Collaborative intelligence between the cloud and mobile edge," *ACM SIGARCH Computer Architecture News*, vol. 45, no. 1, pp. 615–629, 2017.

[31] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2018.

[32] D. C. Nguyen, P. N. Pathirana, M. Ding, and A. Seneviratne, "Privacy-preserved task offloading in mobile blockchain with deep reinforcement learning," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2536–2549, Dec. 2020.

[33] Z. Zhang, F. R. Yu, F. Fu, Q. Yan, and Z. Wang, "Joint offloading and resource allocation in mobile edge computing systems: An actor-critic approach," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, Dec. 2018.

[34] L. Huang, X. Feng, L. Qian, and Y. Wu, "Deep reinforcement learning-based task offloading and resource allocation for mobile edge computing," in *International Conference on Machine Learning and Intelligent Communications*. Springer, 2018, pp. 33–42.

[35] M. Andrychowicz, M. Denil, S. Gomez, M. W. Hoffman, D. Pfau, T. Schaul, B. Shillingford, and N. De Freitas, "Learning to learn by gradient descent by gradient descent," in *Advances in neural information processing systems*, 2016, pp. 3981–3989.

[36] A. Gupta, R. Mendonca, Y. Liu, P. Abbeel, and S. Levine, "Meta-reinforcement learning of structured exploration strategies," in *Advances in Neural Information Processing Systems*, 2018, pp. 5302–5311.

[37] 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 Communications Letters*, vol. 25, no. 5, pp. 1568–1572, may 2021.

[38] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *Proceedings of the 34th International Conference on Machine Learning-Volume 70*. JMLR. org, 2017, pp. 1126–1135.

[39] 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 Transactions on Parallel and Distributed Systems*, vol. 32, no. 1, pp. 242–253, 2021.

[40] 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.

[41] V. Haghighi and N. S. Moayedian, "An offloading strategy in mobile cloud computing considering energy and delay constraints," *IEEE Access*, vol. 6, pp. 11 849–11 861, 2018.

[42] D. Van Le and C.-K. Tham, "A deep reinforcement learning based offloading scheme in ad-hoc mobile clouds," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2018, pp. 760–765.

[43] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 30, no. 1, 2016.

[44] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, 2020.

[45] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

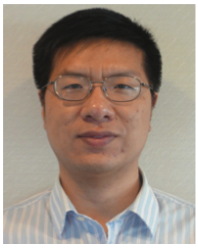
[46] S. Song, Z. Fang, Z. Zhang, C.-L. Chen, and H. Sun, "Semi-online computational offloading by dueling deep-q network for user behavior prediction," *IEEE Access*, vol. 8, pp. 118 192–118 204, 2020.

[47] S. S. Gill, "Quantum and blockchain based serverless edge computing: A vision, model, new trends and future directions," *Internet Technology Letters*, feb 2021.

[48] M. S. Aslanpour, A. N. Toosi, C. Cicconetti, B. Javadi, P. Sbarski, D. Taibi, M. Assuncao, S. S. Gill, R. Gaire, and S. Dustdar, "Serverless edge computing: Vision and challenges," in *2021 Australasian Computer Science Week Multiconference*. ACM, feb 2021.

Guanjin Qu received the bachelor's degree from Taiyuan University of Technology, China in 2019. He is currently working towards the Master's degree at the Center for Applied Mathematics, Tianjin University, China. His research interests include distributed deep learning and edge computing.

Huaming Wu received the B.E. and M.S. degrees from Harbin Institute of Technology, China in 2009 and 2011, respectively, both in electrical engineering. He received the Ph.D. degree with the highest honor in computer science at Freie Universität Berlin, Germany in 2015. He is currently an associate professor at the Center for Applied Mathematics, Tianjin University, China. His research interests include internet of things, wireless and mobile network systems, edge/cloud computing, deep learning and complex networks.



Ruidong Li is an associate professor at Kanazawa University, Japan. Before joining this university, he was a senior researcher at the National Institute of Information and Communications Technology (NICT), Japan. He received the M.Sc. degree and Ph.D. degree in computer science from the University of Tsukuba in 2005 and 2008, respectively. He serves as the secretary of IEEE ComSoc Internet Technical Committee (ITC), and are the founders and chairs of IEEE SIG on Big Data Intelligent Networking and IEEE SIG on Intelligent Internet Edge. He is the associate editor of IEEE Internet of Things Journal, and also served as the guest editors for a set of prestigious magazines, transactions, and journals, such as IEEE communications magazine, IEEE network, IEEE TNSE. He also served as chairs for several conferences and workshops, such as the general co-chair for IEEE MSN 2021, AIVR2019, IEEE INFOCOM 2019/2020/2021 ICCN workshop, TPC co-chair for IWQoS 2021, IEEE MSN 2020, BRAINS 2020, IEEE ICDCS 2019/2020 NMIC workshop, and ICCSSE 2019. 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. He is a senior member of IEEE and a member of IEICE.

Pengfei Jiao received the Ph.D. degree in computer science from Tianjin University, Tianjin, China, in 2018. He is a lecturer with the Center of Biosafety Research and Strategy of Tianjin University. His current research interests include complex network analysis, data mining and graph neural network, and currently working on temporal community detection, link predication, network embedding, recommender systems and applications of statistical network model. He has published more than 50 international journals and conference papers.