IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received March 24, 2020, accepted April 20, 2020, date of publication May 6, 2020, date of current version May 19, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2991773

Distributed Edge Computing Offloading Algorithm Based on Deep Reinforcement Learning

YUNZHAO LI[1], FENG QI[1], ZHILI WANG[1], XIUMING YU[2], AND SUJIE SHAO[1]
[1]State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, China
[2]China Electronics Standardization Institute, Beijing 100007, China

Corresponding authors: Zhili Wang (zlwang@bupt.edu.cn) and Xiuming Yu (yuxiuming@cesi.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2019YFB2102302, in part by the Beijing Natural Science Foundation under Grant 4194085, and in part by the Construction of Industrial Internet Platform Test Bed (New Mode).

ABSTRACT As a mode of processing task request, edge computing paradigm can reduce task delay and effectively alleviate network congestion caused by the proliferation of Internet of things(IoT) devices compared with cloud computing. However, in the actual construction of the network, there are various edge autonomous subnets in the adjacent areas, which leads to the possibility of unbalance of server load among autonomous subnets during the peak period of task request. In this paper, a deep reinforcement learning algorithm is proposed to solve the complex computation offloading problem for the heterogeneous Edge Computing Server(ECS) collaborative computing. The problem is solved based on the real-time state of the network and the attributes of the task, which adopts Actor Critic and Policy Gradient's Deep Deterministic Policy Gradient(DDPG) to make optimized decisions of computation offloading. Considering multi-task, the heterogeneity of edge subnet and mobility of edge devices, the proposed algorithm can learn the network environment and generate the computation offloading decision to minimize the task delay.The simulation results show that the proposed DDPG-based algorithm is competitive compared with the Deep Q Network(DQN) algorithm and Asynchronous Advantage Actor-Critic(A3C) algorithm. Moreover, the optimal solutions are leveraged to analyze the influence of edge network parameters on task delay.

INDEX TERMS Edge computing, computation offload, collaborative computing, reinforcement learning, DDPG.

I. INTRODUCTION

IN order to meet the access network requirements of different devices in different scenarios [1], network operators use heterogeneous access methods (wired or wireless access point) when they build networks. Limited by the computing capability of the IoT device itself, the IoT device will offload some tasks to the cloud server to reduce the device's own burden. However, with the advent of the 5G era, the number of connected devices will be exploded, and a large number of tasks from connected IoT devices will squeeze bandwidth and cause network congestion. In addition, the long distance between the cloud server and the IoT device has brought about severe time delay in the cloud computing mode.

Aiming at the above problems of the cloud computing model and meeting the requirements of network devices for low latency, low energy consumption, and better quality for network services, academia and industry have begun to conduct in-depth research on new computing modes, and proposed paradigms of edge computing types such as mobile edge computing(MEC), fog computing [2], and cloudlet [3]. Edge computing is a shared application mode with communication, computing, and storage functions. Compared with cloud computing, edge computing mode is closer to connected devices in geography, which can effectively solve the problems of time extension and network congestion in cloud computing mode [4], [5]. However, compared to cloud servers, the resources and service scope of a single edge server are limited. When excessive edge devices offload tasks to an edge server, the entire edge network resources will be underutilized.

The associate editor coordinating the review of this manuscript and approving it for publication was Chuan Huang.

In order to alleviate the burden on a single edge server and effectively utilize the computing resources of the whole edge network, a distributed MEC model was proposed [6]–[8] to coordinate global edge servers for distributed computing. Without loss of generality, we name the edge server connected by Edge Device (ED) as access Edge Computing Server (aECS) and the adjacent edge server as collaborate Edge Computing Server (cECS). In distributed MEC mode, aECS assigns part of the received task to the nearby cECS through its associated coordinator. In this way, distributed MEC makes each edge server responsible for part of the task in order to effectively use edge network resources.

However, the existed modes become complicated due to the collaboration between edge servers as well as mobility of services. In heterogeneous network scenarios, offload decisions in the distributed MEC mode becomes very complicated. That is because the offload decisions include task division between ED and aECS, aECS and cECS, as well as device mobility in wireless scenarios. It is difficult to handle such a complex decision problem by traditional optimization algorithms. Fortunately, the revolution in artificial intelligence technology triggered by the rise of deep learning has brought new ideas to solve complex model problems. In solving high-dimensional state and action space decision problems, the deep reinforcement learning (DRL) algorithm [9]–[11] has been proven to be more effective than the traditional methods. Therefore, deep reinforcement learning is adopted to solve the above computation offloading and services migration issues in collaborative edge computing.

In this study, edge servers from an overall perspective of heterogeneous edge subnets are selected to solve the multi-user edge computing offloading problem by using distributed computing manner. Based on DDPG, a distributed computing algorithm is proposed, by which different types of tasks can be offloaded to appropriate locations in the edge subnet for execution. Simulation studies will be carried out to compare with traditional computation offloading methods in heterogeneous scenarios [12]. The contributions of this article are as follows:

1) The computation offloading model is proposed in heterogeneous network scenarios. Different from the existing edge computing offloading model in a single scenario, the proposed model can effectively utilize the computing resources among heterogeneous subnets, and maximize resource utilization in heterogeneous scenarios.
2) The resource allocation algorithm based on reinforcement learning with Task completion rate(TCR) and total task time delay is proposed. Compared with pure consideration of delay, TCR and delay can better offload different types of tasks to different site of network, and more effectively use broadband and computing resources.
3) The performance of the proposed solution is compared with the DQN algorithm. In addition, DQN and other else offloading strategies are used to analyze the impact of computing resource allocation on offloading decision calculation.

II. RELATED WORK

Edge computing is proposed for edge access network with distributed cloud computing capabilities. Compared with cloud computing mode, it has lower latency and higher computing capabilities [13]. However, the computing capability of a single MEC server has not been able to meet the challenge of the growing number of connected devices. Thus, computing offloading and resource allocation have become hot issues in edge computing [14]–[16]. Traditional resource allocation and calculation offloading are solved by clustering or convex optimization methods [17]–[21]. [17] designed a micro-cloud computing (CLOUDLET) in an Orthogonal Frequency Division Multiplexing Access (OFDMA) system with multiple mobile devices, which can simultaneously allocate radio resources and computing resources. The work in [18] used time division multiple access (TDMA) at the communication end to download the results in a predetermined time slot. The literature [19] Obtained the optimal data allocation scheme based on particle swarm optimization of simulated annealing. Unlike the above three studies, which are optimized from the perspective of download links, Authors in [20] optimized the mission uplinks and achieved satisfactory results. The above method is optimized from the perspective of traditional communication. The above researchers have used traditional optimization methods to optimize the allocation of channel resources and computing resources, and have achieved certain results.

In recent years, researchers have tried to optimize the offloading and resource allocation with different optimization objectives. The method was proposed in [21] generated considering both cost and performance factors, an effective resource allocation scheme can be generated. A resource allocation and provisioning algorithm was proposed in [22] by using resource ranking and provision of resources in a hybrid and hierarchical fashion. Reference [23] proposed a Support and Confidence based (SCB) technique which optimises the resource usage in the resource monitoring service. However, as the network scale becomes increasingly complex, there are solutions which are better than the traditional methods.

In order to achieve better edge network resource utilization, the tasks are divided into subtasks in a distributed computing manner and handed over to multiple edge servers for processing into a computing offload mode [24]–[26]. The literature [24] utilize specific repetitive structures of computation allocation at the user to provide coding opportunities that reduce the shuffling load by a factor that increases linearly with the number of users. The work in [25] developed the ''Stackelberg'' game to model the interaction between the edge cloud and users, where the edge cloud sets prices based on limited computing capability to maximize its revenue. For a given price, each user makes offload decisions locally to minimize their own costs. Authors in [26] a decentralized

edge cloud infrastructure that explores the use of voluntary resources for computing and data storage.

At present, most of the research on computation offloading is in a single scenario. With the rollout of 5G network construction, multiple access methods and edge subnets are possible in an area. Several studies on multi-access edge computing have been considered [27]–[29]. Authors in [27] considered a MIMO multi-cell system in which multiple mobile users (MUs) required the transfer of computing tasks to a public cloud server. The literature [28] decomposed the original problem into a resource allocation (RA) problem with fixed task offloading decisions and a task offloading (TO) problem that optimizes the optimal value function corresponding to the RA problem, solve the RA problem by using convex optimization manner, and a novel heuristic algorithm is proposed for the TO problem. This algorithm achieves a suboptimal solution in polynomial time. Software defined network (SDN) are used in heterogeneous edge network. Reference [30] proposed a heterogeneous vehicular networks by using the SDN. Reference [31] propose a software-defined adaptive transmission control protocol (SATCP) for selecting various transmission control policies to adapt to the time-varying vehicular environment. Reference [32] propose a SINET customized solution enabling crowd collaborations for software defined vehicular networks. Reference [33] propose a joint resource allocation and task scheduling approach to efficiently allocate the computing resources to virtual machines and schedule the offloaded tasks. The work in [29] proposed an optimization framework for computation offloading and resource allocation for multi-server mobile edge computing system. The framework aims to minimize system-wide computing overhead by jointly optimizing each computing decisions, transmission capabilities, and computing resources on the server.

In recent years, some studies related to resource allocation and computation offloading use DRL as a solution [34]–[38]. The literature [34] started with finite blocklength code, combined with Deep q-learning algorithm for resource allocation, reduced delay violation rate, and achieved better results than random and equal scheduling benchmark. Reference [35] used the deep reinforcement learning method to make data migration decisions for multi-access edge computing in a dynamic network environment. Reference [36] used DDQN on virtual edge computing and obtained good performance in offloading computing. The literature [37] applied the Monte Carlo tree search algorithm to the resource allocation of MEC, and the performance of the scheme was significantly better than that of DQN. Reference [38] combined the computing offload of edge computing with block chain, comprehensively considered the time delay energy consumption and the cost of the block chain, and achieved a good effect with the DRL-based algorithm.

It can be seen from the simulation results of the above article that the DRL-based algorithm performs better than traditional optimization algorithms. In the field of DRL, we find that the algorithm based on Q-learning cannot reach the optimal due to its discrete actions. So the Actor-Critic-based DDPG algorithm is used to optimize the strategy of this paper. Different from previous research, this paper comprehensively considers distributed computing offloading method and DDPG algorithm to optimize edge computing in heterogeneous edge network scenarios.

III. SYSTEM MODEL

A. THE SYSTEM STRUCTURE

Fig. 1 show the 3 level architecture of the heterogeneous edge network. From bottom to top are IoT device layer, heterogeneous edge computing layer and cloud computing layer. Among of them, the heterogeneous physical layer is composed of different types of edge devices, such as driverless cars, smart homes, smart phones, etc. In this architecture, the networking scenarios (such as wired and wireless network scenes) where edge devices located are different. The characteristics of the tasks (such as traffic characteristics and time delay characteristics) to be performed on the device are also different. Users access the network in different ways through various IoT devices. These devices generate tasks and send requests to edge subnets which they connect. Agents of autonomous edge subnets will get the network state of other subnets through distributed file system. After receiving the task request from the Internet of things device, the agent makes the decision of computing and offloading according to the characteristics of the request. After training, computation intensive tasks tend to be executed on cloud servers, while time sensitive task tend to be performed on local and edge servers.

B. HETEROGENEOUS NETWORK MODEL

In the proposed Structure, the subnets of the edge layer are not only autonomous but also heterogeneous. These edge subnets may be composed in different ways. As described in Fig. 1. In order to meet the requirement for different types of tasks, an edge device will be covered by multiple heterogeneous edge subnets at the same time with its own access mode. Edge subnets with different structures have different characteristics. For example, a wireless subnet composed of interconnected wireless access points (AP) can handle mobile edge devices well and has minimal requirements for the infrastructure of the subnet, while wired edge subnets have the most abundant communication resources and are suitable for tasks with heavy traffic characteristics. Wireless edge subnet has the best mobility, fiber wireless hybrid edge subnet has both mobility and high bandwidth and low latency, wired edge subnet has the highest bandwidth and the lowest latency, but it cannot be suitable for moving edge equipment. Edge device selects different subnet computing offloading according to the task properties.

In each subnet, an edge node that is fully connected to other ECS is designated as the coordinator, which is responsible for collecting task offload requests received by each ECS, making offloading decisions, and coordinating the computing

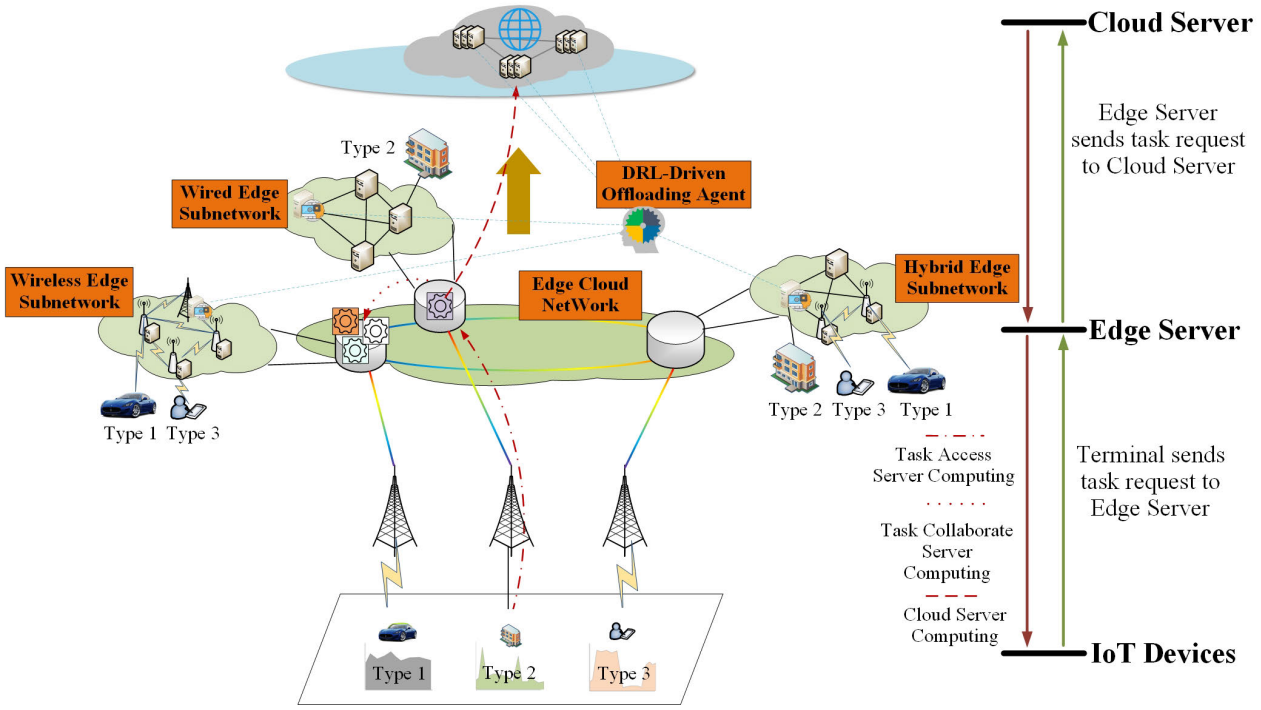


FIGURE 1. Three-layer edge network structure.

resources of this subnet and other subnets to maximize Edge computing resources for each subnet.

C. WORKFLOW OF THE COORDINATOR'S WORK

As shown in Fig. 2, the decision workflow of computing offloading based on reinforcement learning can be divided into two parts: information input flow and decision output flow. Information input flow is the information flow from edge network to decision optimization engine based on reinforcement learning. The decision output flow is the decision control flow from the decision optimization engine to the edge network entity.

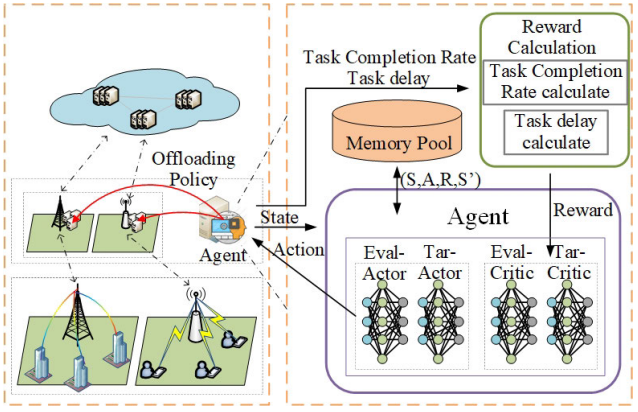


FIGURE 2. Process of heterogeneous edge network distributed computing offload Algorithm.

The edge device accesses the edge network through different access methods and requests to offload the task upward. At this time, the service request coordinator obtains the task request from the edge device, then extracts the feature information of the task, abstracts it into a feature vector suitable for the task feature of the decision engine, and inputs the decision optimization engine. When Ed generates tasks and sends service requests to the network, the requests which consist of the task's eigenvectors will be captured by agents in the subnet. In the network architecture proposed in this paper, the coordination between edge servers is realized by distributed file system. Through the distributed file system, agents in each edge subnet can obtain the real-time network state. Because of this, the agents can make the right decision for computing offload. Agent in the edge network offload tasks to the appropriate location in the network for processing, so as to achieve the overall performance optimization.

After obtaining the network state matrix and task eigenvector, the decision optimization engine makes the decision of computation offloading and deploys the physical instance through the edge network agent. In section V, there are the further explanation of decision algorithm with Fig. 2.

D. HETEROGENEOUS EDGE NETWORK COMPUTING OFFLOADING MODEL

The edge network consists of N_D edge devices, each of which is connected to a different ECS according to its communication characteristics. Limited by the weak computing capability of the edge device itself, the task execution time will be very long for computation-intensive tasks if they

are all performed locally. Therefore, it is a good choice to offload tasks to the edge server in a reasonably way. In this process, each ECS will carry a large number of tasks from edge devices. We record the task waiting queue of edge server e_i as q_{e_i}, which indicates that there are tasks with q_{e_i} time to be processed. If the q of ECS is very long, it means that the ECS is overloaded, and the task will be offloaded to the cloud server for processing.

Different types of edge devices tend to different task characteristics, such as different network access methods, different traffic characteristics, and different time delay requirements. In order to meet different types of tasks, heterogeneous edge network is needed to meet different task requirements. In this paper, access networks are chose to access wireless edge network by identifying the edge of the device type, such as smart car, smart phones, which have the strongest mobility. Sweeper robots with certain mobility and large data flow access conditions have priority access to wireless fiber hybrid edge subnet, while devices with large task data flow and optical fiber access conditions, such as smart TV, have priority access to wired edge subnet.

In this framework, tasks are distributed to local, edge and cloud servers respectively according to the strategy. According to the network state and task characteristics, aECS assigns tasks to the adjacent ECSs for collaborative processing. If the adjacent edge servers also have no spare computing capability, the tasks is sent to the cloud computing center for execution. When the adjacent cECS finishes the task, the task will be returned to access aECS, which will be integrated and returned to IoTD.

IV. PROBLEM FORMULATION

In this section, the optimization problem will be formulated with two objectives. The goal of this paper is to find a computing offloading strategy with expected minimum task delay under the condition of ensuring TCR. The task delay is determined by the maximum value of the local execution and the offloading computation.

A. NETWORK AND TASK STATUS AND CHARACTERISTICS

The state of each episode can be expressed as a state set $S(S^E, T^d, S^d, L^{E\to E})$, as explained below.

- S^E represents the state set of the edge server, $S^E = \{S_j = (q_{e_j}, f_{e_j})|j = 1, 2, \ldots, m\}$, q_{e_j} is the task queue duration of the ECS e_j, and f_{e_j} is the frequency of the ECS e_j CPU. Where q_{e_j} is given as follows:

$$q_{e_j} = \sum_{i=1}^{N} \frac{c_i \times a_i^{d\to e}}{f_{e_j}} \tag{1}$$

- T^d represents the feature set of the task, $T^d = \{c_i, td_i, s_i^u, s_i^d | i = 1, 2, \ldots, N\}$, c_i represents the computing cycle required to process task i, td_i is the Deadline of task i, s_i^t is the size of task.
- S^d indicates the state and characteristics of the edge device, $S^d = \{S_j = (q_{d_j}, f_{d_j})|j = 1, 2, \ldots, n\}$, q_{d_j} is the task queue duration of edge device d_j, and f_{d_j} is the CPU frequency of edge device d_j. Where q_{d_j} is given by

$$q_{d_j} = \sum_{i=1}^{N} \frac{c_i \times a_i^d}{f_{d_j}} \tag{2}$$

- $L^{E\to E}$ represents the communication link bandwidth between the nodes.

B. OFFLOAD ACTION

The task offloading action consists of three parts: from ED to aECS, from aECS to cECS, and from aECS to cloud server. The offloaded task includes the task execution code and the task data. It has been found that the offloading ratio of the computing cycle is roughly equal to the total amount of task offload, so we merge the two sub-actions of the ED offload to aECS. Then each episode's set of computation actions can be expressed as $A(A^{d\to\hat{e}}, A^{\hat{e}\to e}, A^{\hat{e}\to\breve{e}})$, as specified below:

- $A^{d\to\hat{e}}$indicates the proportion of task calculation/data amount offloaded from ED to aECS.
- $A^{\hat{e}\to e}$indicates the proportion of task calculation/data amount offloaded from aECS to cECS.
- $A^{\hat{e}\to\breve{e}}$indicates the proportion of task calculation/data amount offloaded from aECS to cloud server.

C. COMPUTING MODEL

1) LOCAL EXECUTION

The local computing time of the task consists of two parts, namely the local execution time and the local task queue waiting time. The total completion time of the local computing task is calculated by the following formula:

$$t^{local} = a_i^d \times \frac{c_i}{f_{d_j}} + q_{d_j} \tag{3}$$

2) SERVER EXECUTION

If the task is all executed locally, the ED computing capability is not sufficient to complete it within the task's deadline, so the ED needs to offload a portion of the task to the edge server. The time spent in the computation offloading consists of task queuing time, task execution time, and task transfer time. The edge calculation time + queuing time + transmission delay is calculated by the following formula.

$$t_i^{cal+que} = \max(a_i^{d\to\hat{e}} \times \frac{c_i}{f_{\hat{e}}} + q_{\hat{e}}, a_i^{\hat{e}\to e_j} \times \frac{c_i}{f_{e_j}} + q_{e_j}, a_i^{\hat{e}\to\breve{e}} \times \frac{c_i}{f_{\breve{e}}} + q_{\breve{e}} | j \in \{1, 2, \ldots, m-1\}) \tag{4}$$

The calculation formula of the transmission delay of the task is as follows.

$$t_i^{trans} = \max(a_i^{d\to\hat{e}} \times \frac{s_t}{l^{d\to\hat{e}}}, a_i^{\hat{e}\to e_j} \times \frac{s_t}{l^{\hat{e}\to e_j}}, a_i^{\hat{e}\to\breve{e}} \times \frac{s_t}{l^{\hat{e}\to\breve{e}}} | j \in \{1, 2, \ldots, m-1\}) \tag{5}$$

3) OPTIMIZATION GOAL

Our optimization objective takes into account the expectation of task delay and TCR. Based on the above formula, the task completion time is

$$t_i = \max(t_i^{local}, (t_i^{cal+que} + t_i^{trans})) \tag{6}$$

We assume that there are P tasks in total, so the expected delay of all tasks is

$$E_t = \frac{\sum_{i=1}^{P} t_i}{P} \tag{7}$$

The purpose of this paper is to maximize the TCR while making the task latency expectation as small as possible. Therefore, when the task is not completed in the deadline, a negative reward is given, and when the task is completed in the deadline, a high positive reward is given. We set the reward as:

$$r_i = \begin{cases} -t_i & t_i > td_i \\ \log_{0.995}(1 - \frac{1}{e^{\sqrt{t_i}}}) & t_i < td_i \end{cases} \tag{8}$$

In order to explain the role of reward more clearly, an example will be proposed as follow. Assuming that the deadline of a task t_i is 8ms, and the execution time of the task is 9ms in a one training. So that the reward generated in this state is -9. In the next iteration training, the execution time of task ti is 7ms, so the reward of the state is 14.68. After this iteration, DRL agent finds that the total rewards of the second iteration is greater than the previous, and actions in this iteration will be remembered by neural network. In the another iteration training, the execution time of task ti is 5ms, and the reward of this state is 22.55, which is greater than the previous two iterations. After continuous iterative training, the decision-making of DRL agent's computing offload will perform more outstanding in general. The meanings of the symbols defined in this article are shown in **Table 1**.

TABLE 1. Symbol summary.

Symbol	Definition
S	The state of edge computing network.
S^E	The state of edge servers.
T^d	The properties of a task.
S^d	State and characteristics of edge devices.
$L^{E\to E}$	Communication link bandwidth between nodes.
N_D	The number of ED
M	The amount of ECS.
q	The number of the task which completed in deadline.
$\hat{e}$	The access edge server.
e	The collaborative edge server.
$\breve{e}$	The cloud server.
q	CPU queue wait time.
td	The deadline of the task.
c	CPU calculation cycle required for the task.
f	CPU frequency.
s^t	The Task data size.
A	The offload action collection.
$A^{d\to\hat{e}}$	The proportion of the calculation cycle/data amount that the task is offloaded from ED to the aECS.
$A^{\hat{e}\to e}$	The proportion of the calculation cycle/data amount that the task is offloaded from aECS to the cECS.
$A^{\hat{e}\to\breve{e}}$	The proportion of the calculation cycle/data amount that the task is offloaded from aECS to the Cloud server.
A^d	The proportion of the calculation cycle/data amount that the task is executed on local.

V. RESOURCE ALLOCATION ALGORITHM BASED ON DRL

In this section, the complex computation offloading problem is abstracted into the MDPs. In the next step, we propose a collaborative computing resource allocation algorithm based on deep reinforcement learning, which is used to solve the optimization problem of offloading action. It consists of four neural networks and an experience pool.

A. MARKOV DECISION PROCESS

The optimization goal t_i is an single time slot object, it depends only on the current state. However, the state of network environment is dynamically, so the past network states are also important reference factors for computation offloading actions. If only the current state is taken to make a decision, the decision action made by agent will lack further vision.

Reinforcement learning is a method to optimize problems in a dynamic environment. We describe the problem as a simple one-slot problem, and then add the past state on this basis. So that, we first formulated the problem as a MDPs, which can be represented by a tuple $\{S, A, P, R\}$, S is the state space of the model, and A is the action space of the model. P is the state transition matrix of the model, and R is the reward function of the model.The description of each element is as follows:

- The state space in this article is defined as the state of each server and task in the edge network. The server set is $S_{ESC} = \{s_1, s_2, \ldots s_n\}$, where n is the number of servers in the edge network. The server status is task waiting queue length, and the task set is A, where v is the total number of tasks. Task status $S_M = \{m_1, m_2, \ldots, m_v\}$, $q_i \in [0, 1]$, $\sum_{i=1}^{n} q_i = 1$ represents the percentage of tasks assigned by the server at various locations.
- At each moment, after considering the task waiting queue length of each node and the deadline of the task itself, the agent must make an action to assign the task to each server for processing. We define the action space as

$$A(A^{d\to\hat{e}}, A^{\hat{e}\to e}, A^{\hat{e}\to\breve{e}})$$

and the constraint as

$$A^{d\to\hat{e}} + A^{\hat{e}\to e} + A^{\hat{e}\to\breve{e}} = 1$$

The meaning of each element is given in Section 3.
- Every time an agent makes an action, the environment automatically gives a reward: here we define the reward value as (8), so the total reward is

$G = \sum_{t=1}^{T} R(s_t', a_t', s_{t+1})$. Our ultimate goal is maximin the total reward.

In addition to the above four elements, there is a hyper-parameter γ. γ is the future reward weight, the value range is [0, 1]. The value function is focused on the currently obtained reward when γ tends to 0. While. The value function will consider more rewards from the followed steps if γ tends to 1. In other words, γ makes decisions biased towards short-term rewards or long-term rewards.

B. DYNAMIC RESOURCE OPTIMIZATION ALGORITHM BASED ON DRL

In the resource allocation decision, we need to directly interact with the environment to get the sample, and through the resulting sample estimate value function, the ultimate goal is to find out the optimal strategyπ_*. The state space and action space of the network model in this paper are characterized by high dimensionality, dynamics, and non-discrete action, which need to be optimized in the process of sequence generation. Based on the above factors, we choose DDPG-based methods to optimize our decision-making algorithms. DDPG is derived from the improved version of Actor-Critic and Policy gradient algorithms, and also draws on the double network structure of DDQN [39].

As shown in the Fig. 2, the deployment of DRL consists of two parts: the network environment and the intelligent agent. The network environment consists of network nodes (edge nodes and cloud nodes), network monitors, and users. The network node receives the task offload request from the user, and the network monitor collects the information in the network in real time and interacts with the intelligent agent information to respond to the state change in the network.

DDPG uses a policy function $\pi_\theta(s)$ to make a decision. It definitely maps a state to a specific action. Compared to a random strategy, its action choice is only one, which greatly improves the convergence of training. In Actor-Critic schema, the agent uses the Policy Gradient method to enhance Gradient, and directly selects the action with the highest probability in the current state through the policy function $\pi_\theta(s)$. Correspondingly, the Critic network evaluates the current decision based on the TDerror between the value function and the current reward, and evaluates the behavior of the Actor. The gradient calculation formula for the deterministic strategy gradient based on Q value is:

$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^\pi}[\nabla_\theta \pi_\theta(s) \nabla_a Q_\pi(s, a)|a = \pi_\theta(s)] \quad (9)$$

As shown in the Fig. 2, the distributed dynamic resource optimization algorithm based on DRL is composed of four parts: edge network environment, experience pool, Actor double network and Critic double network, in which the two networks of Actor and the two networks of Critic have the same structure respectively.

The network control node interacts with the edge network environment to obtain the current network state and stores the current state vector $\phi(S)$, action vector A, reward R, and next state vector $\phi(S')$ to the experience pool. In order to explore the action space more broadly, we add some noise to the action selected by the Actor. Then, after accumulating a certain amount of data in the experience pool, the data block with the size of mini-batch will be taken out and input into the estimated neural network to obtain the action value function. The loss function $\frac{1}{m}\sum_{j=1}^{m}(y_j - Q(\phi(S_j), a_j, \omega))^2$ is calculated together with the action value function of the target value network, and all parameters ω of the current network were updated through the gradient reverse transmission of the neural network.

With the loss function, Eval-Net uses gradient descent to update the parameters in the network. After a certain episode, Eval-Net copies the latest parameters to Target-Net. After several rounds of episode parameter update, the loss function of Eval-Net tends to converge, and after obtaining the trained neural network parametersω, the optimal strategy π_*can be obtained.

First, we define the current target Q value, which is used to calculate the expected reward value of the action in the current state, which is defined as follows:

$$y_j = R_j + \gamma Q_{target}(\phi(S_j'), \pi_{\theta'}(\phi(S_j')), \omega') \quad (10)$$

This represents the weighted expectation of the current state of reward and possible future rewards, and is used to evaluate the value of the current state.

Next, we define how the Actor-Critic neural network parameters are updated:

$$\omega' \leftarrow \tau\omega + (1-\tau)\omega' \quad (11)$$

$$\theta' \leftarrow \tau\theta + (1-\tau)\theta' \quad (12)$$

Unlike DQN, which directly copies the parameters of the target-network to the eval-network, this algorithm uses a gradual update method, and each parameter is updated only by a small amount. At the same time, in order to increase the randomness of the learning process and better explore the entire solution space, we add some noise to the learning process. The action selection expression is defined as follows:

$$A = \pi_\theta(S) + \eta \quad (13)$$

where η is noise. Next, we define the loss function of the Critic network and the Actor network. Critic network's loss function is defined as follows:

$$J(\omega) = \frac{1}{m}\sum_{j=1}^{m}(y_j - Q(\phi(S_j), a_j, \omega))^2 \quad (14)$$

For the loss function of the Actor network, refer to (15). The loss gradient defined in the article [39] is as follows:

$$\nabla_\theta J(\pi_\theta) = E_{s\sim\rho^\pi}[\nabla_a Q_\pi(s, a)|_{s=s_i, a=\pi_\theta(s)} \nabla_\theta \pi_\theta(S)|_{s=s_i}] \quad (15)$$

The computing offloading algorithm based on DDPG is as described in Algorithm 1. It consists of two parts: the initialization of the network environment and the DRL algorithm in the intelligent agent.

Algorithm 1 Computation Offloading in Heterogeneity Edge Environment

Input: environment parameters

Initialize: the actor eval-network and tar-network with parameter θ, θ';
the critic eval-network and tar-network with parameter ω, ω';

for *episode* in range(*Max_Iteration*) **do**
  Initialize edge network state and task queue.
  for each *stept* $= 0, 1, 2 \ldots$ **do**
    Get the current state s_t from edge network environment and transform it to vector $\phi(s_t)$.
    Offload the request according to policy $\pi_\theta(\phi(s))+\aleph$ in actor network;
    Receive reward r_t by (8);
    Store the tuple $\{\phi(s_t), a_t, r_t, \phi(s_{t+1})\}$ to the reply memory D;
    Get the next state s_{t+1};
    Calculate the Q value y_j by(10);
    Update critic by minimizing (14);
    Update the actor policy using the (15);
    Update the Actor tar-network by (11);
    Update the Critic tar-network by (12);
    t=t+1;
  end for
end for

Output: Optimal Actor current network parameter θ, Critic current network parameter ω

At the beginning, the decision made by the intelligent agent is close to the random algorithm. With the learning process of the DDPG-based computing offload algorithm, the obtained offload strategy is getting closer to the optimal algorithm. After the learning iteration ends, the learned DDPG neural network parameters are obtained.

VI. EXPERIMENTAL PERFORMANCE ANALYSIS AND COMPARISON

A. HETEROGENEOUS NETWORK ENVIRONMENT

Because the real edge computing environment is not realized for us to the algorithm, we use the network package Networkx based on Python to build the network environment, and use tensorflow to implement the proposed algorithm. In this section, we perform simulation experiments on DDPG-based computing offloading algorithms in heterogeneous scenarios. In the standard configuration, three edge devices are connected in each scenario. In Fig. 3, the wired scenarios, edge devices are connected to the edge network through optical fibers. In this scenario, we only focus on computing offload, not considering service migration.

In the wireless access scenario, as shown in Fig. 3, we divide the scenario into two categories, one is pedestrian walking, and the other is vehicle movement. Pedestrians move in a low-speed environment, and vehicles move at a constant speed (0-50m/s) in a fixed direction. In the wireless access scenario, the device has mobility, so this scenario is a mobile edge computing scenario. In this scenario, we use the computing offloading scheme under MEC.

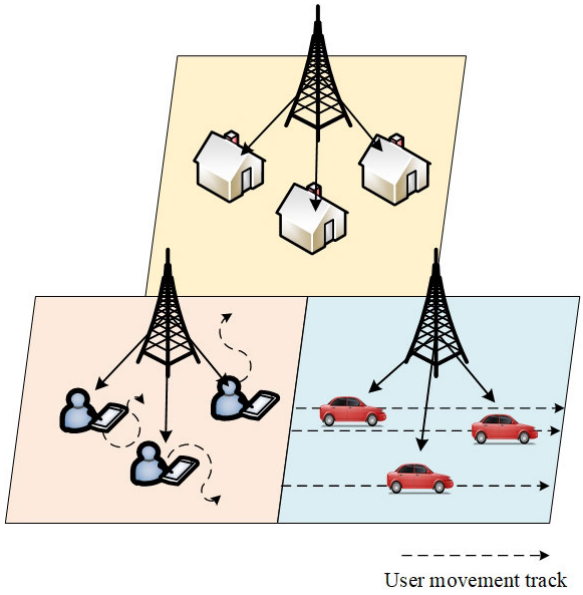


FIGURE 3. Simulation of different access scenarios.

B. SIMULATION SETTINGS

The environment of this simulation is a three-layer edge network, including a terminal layer, a heterogeneous edge layer, and a cloud layer. There are 2-8 wireless edge servers in the edge layer and 1-4 wired edge servers. The ED with fiber access mode has no mobility. The service range of each wireless AP is 50×50m, and the speed of each wireless mobile ED is randomly 0-40m/s. Initially, there are edge devices in each AP service range, and the probability that each edge generates an offload request in each time slot meets the Poisson distribution.

The related parameters in the experiment are listed in **Table 2**, which are applied in simulation examples unless otherwise stated.

TABLE 2. Simulation environment parameters.

Parameter	Description	Values
s^t	Data size of task	100KB-2MB
td	Deadline of task.	5ms-50ms
c	CPU calculation cycle for the task.	0.1
f_d	CPU frequency of ED.	0.5GHZ
f_e	CPU frequency of ECS.	3-5GHZ
$f_{\breve{e}}$	CPU frequency of cloud server.	10GHZ
M	Amount of ECS.	3-12
N_D	Amount of ED.	9
$L^{d\to e}$	The bandwidth from ED to ECS.	50MB/S
$L^{e\to e}$	The bandwidth from to and ECS.	300MB/S
$L^{e\to \breve{e}}$	The bandwidth from ECS to cloud.	200MB/S

DDPG's Actor network and Critic network are three-layer structure, and the second layer of the fully connected layer is composed of 200 neurons. In the first layer, the input state vector is normalized. The hyperparameters of other neural networks are shown in **Table 3**.

TABLE 3. Simulation neural network hyperparameter.

Parameter	Description	Values
$episode$	Number of iterations	6000
a_A	Learning rate of Actor Network.	1×10^{-3}
a_C	Learning rate of Critic Network.	2×10^{-3}
Mem	Size of Experience pool.	10000
$Batch$	Size of Mini-batch.	64
γ	Reward attenuation rate.	0.9
τ	Soft mode parameter update rate.	1×10^{-2}

C. COMPARISON OF ALGORITHMS

In order to study what factors affect the performance of the algorithm, 4 setting are changed in this experiment. Otherwise, there are other 4 offloading method are compared with the proposed algorithm in each scene. The three schemes are as follows.

1) A3C-based offload algorithm: A3C and DDPG are both actor-critic algorithms. It have unique advantages in some aspects, such as the convergence of algorithm training. This algorithm has been used in many studies recently, and that is the reason why we chose this algorithm as a comparison algorithm for experiments.
2) DQN-based offload algorithm: Since the action space of DQN can only be discrete values, we set the action on each dimension of DQN to be 0.2. The IoT device performs computation offloading for each task request according to the decision given by the trained DQN network.
3) Edge server computing: The IoT terminal ignores the burden of the edge server and always offloads the task to the access edge server for calculation.
4) Local computing: The IoT terminal puts all computing work locally and does not request the computing offload to the edge network.

D. SIMULATION RESULTS

In this section, we first observe the convergence of the algorithm in the training process, and then compare it with the performance of other algorithms in the heterogeneous network scenario.

As shown in Fig. 4 and Fig. 5, during the first 3300 rounds of training, the average task delay decreases rapidly. When the number of training reaches 3,500, the task delay tends to be stable. In the first 3,500 rounds of training, the task completion rate increased rapidly, and the training effect converges to 98%-99% after the 3500th.

As depicted in the Fig. 6(a) and Fig. 6(b), the proposed algorithm is compared with other algorithms in different settings of edge network environment. CPU Capacity and

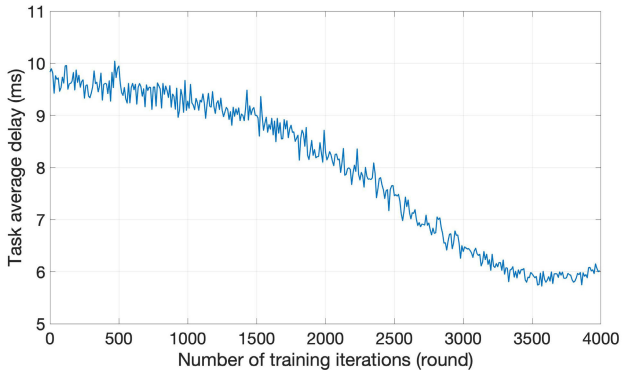


FIGURE 4. Relation between delay with training iterations.

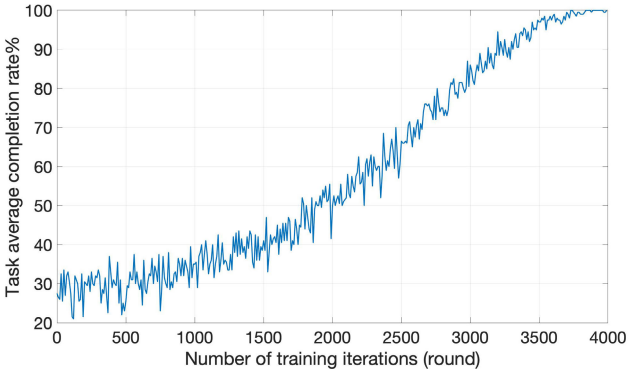
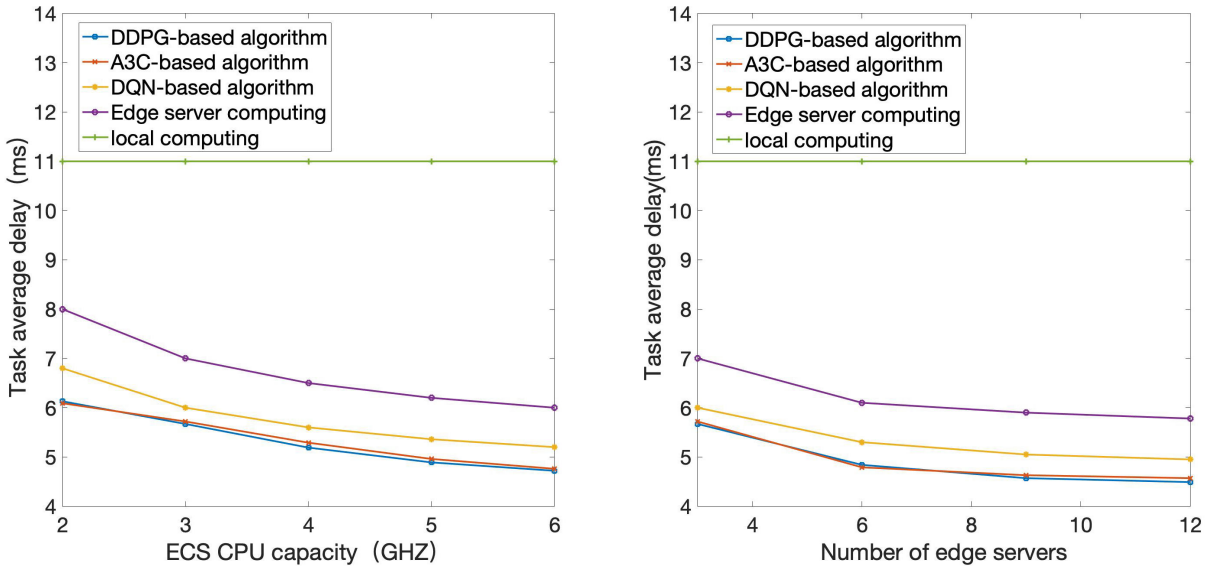


FIGURE 5. Relation between task completion rate with training iterations.

Number of ECS are changed in Fig. 6(a) and Fig. 6(b). After 4000 iterations training, it is 0.8%, 10.9%, 30.5% lower in DDPG-based algorithm when compared with the A3C, DQN, Edge server computing and local computing. We can clearly see that the proposed algorithm and A3C-based algorithm performed significant better then the others. As both DDPG and A3C algorithms belong to actor-critic mode, their performances in each setting are very close. We set the capacity of each edge server as 2GHZ, 3GHZ, 4GHZ, 5GHZ, 6GHZ. When the number of edge servers increases from 3 to 12, the average task delay of other computing methods except local computing is gradually decreasing. After the number of servers is increased to 6, increasing the number of edge servers does not significantly improve the effect. It can be seen that with the improvement of the CPU capacity of the edge server, the average delay of tasks is decreasing. However, this kind of improvement is not obvious compared with the improvement of CPU capacity. We find that task delay is limited by the network bandwidth and the capacity of the terminal's own CPU, and the it has a theoretical limit.

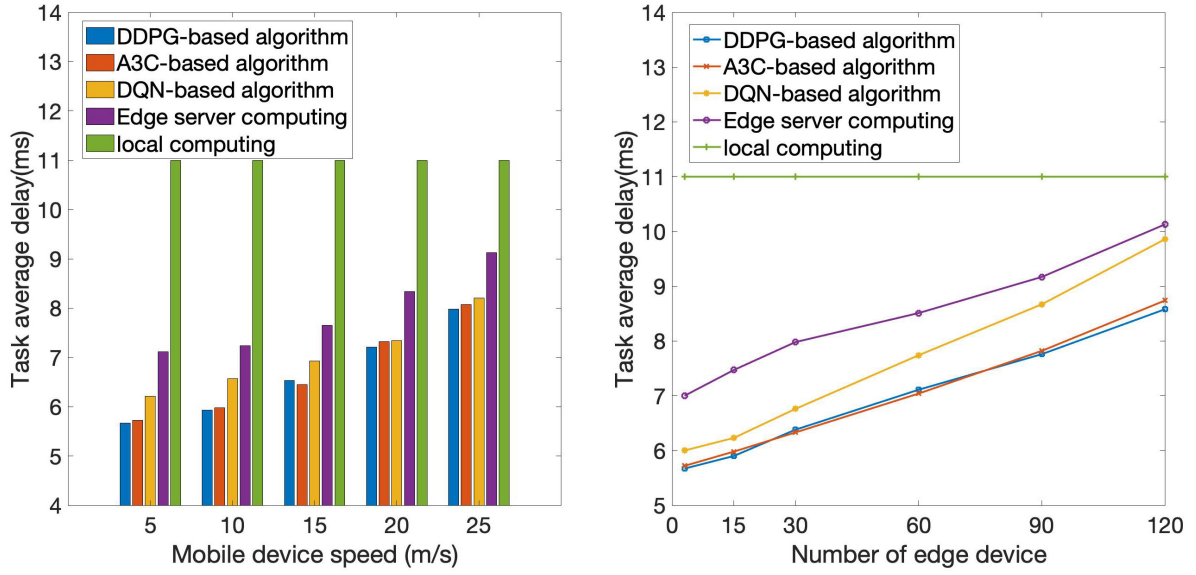
In the MEC scenario, the mobile ED moves from the service area of one AP to the service area of another AP. Due to the migration, the moving speed of the mobile ED also affects the performance in this scenario. According to Fig. 7(a), in addition to the local execution of the ED, as the ED moves faster, the task latency expectation is increasing.



(a) Server computing capacity

(b) Number of edge servers

FIGURE 6. Task delay with different ES computing capacity and number.



(a) Mobile device speed

(b) number of edge devices

FIGURE 7. Task delay with different device speed and number.

Under low speed conditions, the performance based on DDPG algorithm is significantly better than the rest of the offloading mode. As the speed increases to 40m/s, the performance based on DDPG algorithm is only slightly stronger than DQN and A3C.

The impact of the number of mobile terminals on task delay expectations is shown in Fig. 7(b). It can be seen that the algorithm based on DDPG performs optimally, and as the number of mobile terminals increases, the task delay expectation increases. In the mobile edge computing scenario, the increase of number of the mobile terminal brings about an increase in the number of task requests in the time slot, which increases the burden on the edge server.

VII. CONCLUSION

To satisfy three heterogeneous edge networks using one offloading algorithm, this paper combines three heterogeneous edge networks with remote cloud networks, and builds a three-layer edge cloud network with cloud-side collaboration. The heterogeneous edge network is used to implement the multi-access capability on the edge side, and the task completion and task delay are used as reward values. Thus, the

trained allocation strategy can reasonably offload different types of tasks to different network locations to achieve the highest task completion rate. At the same time, task delay is minimized. It is verified by simulation studies that the DDPG-based algorithm designed in this study can effectively improve the system performance, reduce the task delay by up to 50% while that most tasks are completed within the deadline. However, the ECS divides tasks in terms of task delay and deadline in this study, we will consider the cost consumption in the future work.

REFERENCES

[1] F. Jalali, T. Lynar, O. J. Smith, R. R. Kolluri, C. V. Hardgrove, N. Waywood, and F. Suits, "Dynamic edge fabric EnvironmenT: Seamless and automatic switching among resources at the edge of IoT network and cloud," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2019, pp. 77–86.

[2] R. K. Naha, S. Garg, D. Georgakopoulos, P. P. Jayaraman, L. Gao, Y. Xiang, and R. Ranjan, "Fog computing: Survey of trends, architectures, requirements, and research directions," *IEEE Access*, vol. 6, pp. 47980–48009, 2018.

[3] K. Gai, M. Qiu, H. Zhao, L. Tao, and Z. Zong, "Dynamic energy-aware cloudlet-based mobile cloud computing model for green computing," *J. Netw. Comput. Appl.*, vol. 59, pp. 46–54, Jan. 2016.

[4] W. Yu, F. Liang, X. He, W. Grant Hatcher, C. Lu, J. Lin, and X. Yang, "A survey on the edge computing for the Internet of Things," *IEEE Access*, vol. 6, pp. 6900–6919, 2018.

[5] M. Caprolu, R. Di Pietro, F. Lombardi, and S. Raponi, "Edge computing perspectives: Architectures, technologies, and open security issues," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2019, pp. 116–123.

[6] A. Khakimov, A. Muthanna, and M. Saleh Ali Muthanna, "Study of fog computing structure," in *Proc. IEEE Conf. Russian Young Researchers Electr. Electron. Eng. (EIConRus)*, Feb. 2018, pp. 51–54.

[7] E. Balevi and R. D. Gitlin, "A clustering algorithm that maximizes throughput in 5G heterogeneous F-RAN networks," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[8] D. Chen and V. Kuehn, "Weighted max-min fairness oriented load-balancing and clustering for multicast cache-enabled F-RAN," in *Proc. 9th Int. Symp. Turbo Codes Iterative Inf. Process. (ISTC)*, Sep. 2016, pp. 395–399.

[9] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proc. 30th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[10] X. Zhao, K. Yang, Q. Chen, D. Peng, H. Jiang, X. Xu, and X. Shuang, "Deep learning based mobile data offloading in mobile edge computing systems," *Future Gener. Comput. Syst.*, vol. 99, pp. 346–355, Oct. 2019.

[11] Z. Ali, L. Jiao, T. Baker, G. Abbas, Z. H. Abbas, and S. Khaf, "A deep learning approach for energy efficient computational offloading in mobile edge computing," *IEEE Access*, vol. 7, pp. 149623–149633, 2019.

[12] S. Li, Y. Tao, X. Qin, L. Liu, Z. Zhang, and P. Zhang, "Energy-aware mobile edge computation offloading for IoT over heterogenous networks," *IEEE Access*, vol. 7, pp. 13092–13105, 2019.

[13] H. Mueller, S. V. Gogouvitis, H. Haitof, A. Seitz, and B. Bruegge, "Continuous computing from cloud to edge," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 97–98.

[14] T. Mengistu, A. Alahmadi, A. Albuali, Y. Alsenani, and D. Che, "A 'no data center' solution to cloud computing," in *Proc. IEEE 10th Int. Conf. Cloud Comput. (CLOUD)*, Jun. 2017, pp. 714–717.

[15] T.-Y. Kan, Y. Chiang, and H.-Y. Wei, "Task offloading and resource allocation in mobile-edge computing system," in *Proc. 27th Wireless Opt. Commun. Conf. (WOCC)*, Apr. 2018, pp. 1–4.

[16] X. Wei, S. Wang, A. Zhou, J. Xu, S. Su, S. Kumar, and F. Yang, "MVR: An architecture for computation offloading in mobile edge computing," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jun. 2017, pp. 232–235.

[17] Y. Yu, J. Zhang, and K. B. Letaief, "Joint subcarrier and CPU time allocation for mobile edge computing," in *Proc. IEEE Global Commun. Conf. (GLOBECOM)*, Dec. 2016, pp. 1–6.

[18] H. Xing, L. Liu, J. Xu, and A. Nallanathan, "Joint task assignment and wireless resource allocation for cooperative mobile-edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[19] S. Zhu, L. Gui, J. Chen, Q. Zhang, and N. Zhang, "Cooperative computation offloading for UAVs: A joint radio and computing resource allocation approach," in *Proc. IEEE Int. Conf. Edge Comput. (EDGE)*, Jul. 2018, pp. 74–79.

[20] L. Ruan, S. Guo, H. Rutagemwa, B. Rong, X. Qiu, and W. Li, "The re-expanded cloud: Distributed uplink offloading for mobile edge computing," in *Proc. IEEE Int. Conf. Commun. (ICC)*, May 2018, pp. 1–6.

[21] S. K. Battula, S. Garg, R. K. Naha, P. Thulasiraman, and R. Thulasiram, "A micro-level compensation-based cost model for resource allocation in a fog environment," *Sensors*, vol. 19, no. 13, p. 2954, 2019.

[22] R. K. Naha, S. Garg, A. Chan, and S. K. Battula, "Deadline-based dynamic resource allocation and provisioning algorithms in fog-cloud environment," *Future Gener. Comput. Syst.*, vol. 104, pp. 131–141, Mar. 2020.

[23] S. K. Battula, S. Garg, J. Montgomery, and B. H. Kang, "An efficient resource monitoring service for fog computing environments," *IEEE Trans. Services Comput.*, early access, Dec. 27, 2019, doi: 10.1109/TSC.2019.2962682.

[24] S. Li, Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, "Poster abstract: A scalable coded computing framework for edge-facilitated wireless distributed computing," in *Proc. IEEE/ACM Symp. Edge Comput. (SEC)*, Oct. 2016, pp. 79–80.

[25] M. Liu and Y. Liu, "Price-based distributed offloading for mobile-edge computing with computation capacity constraints," *IEEE Wireless Commun. Lett.*, vol. 7, no. 3, pp. 420–423, Jun. 2018.

[26] A. Jonathan, M. Ryden, K. Oh, A. Chandra, and J. Weissman, "Nebula: Distributed edge cloud for data intensive computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3229–3242, Nov. 2017.

[27] S. Sardellitti, G. Scutari, and S. Barbarossa, "Joint optimization of radio and computational resources for multicell mobile-edge computing," *IEEE Trans. Signal Inf. Process. Over Netw.*, vol. 1, no. 2, pp. 89–103, Jun. 2015.

[28] T. X. Tran and D. Pompili, "Joint task offloading and resource allocation for multi-server mobile-edge computing networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 1, pp. 856–868, Jan. 2019.

[29] Q.-V. Pham, T. Leanh, N. H. Tran, B. J. Park, and C. S. Hong, "Decentralized computation offloading and resource allocation for mobile-edge computing: A matching game approach," *IEEE Access*, vol. 6, pp. 75868–75885, 2018.

[30] W. Quan, K. Wang, Y. Liu, N. Cheng, H. Zhang, and X. S. Shen, "Software-defined collaborative offloading for heterogeneous vehicular networks," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–9, Apr. 2018.

[31] W. Quan, N. Cheng, M. Qin, H. Zhang, H. A. Chan, and X. Shen, "Adaptive transmission control for software defined vehicular networks," *IEEE Wireless Commun. Lett.*, vol. 8, no. 3, pp. 653–656, Jun. 2019.

[32] W. Quan, Y. Liu, H. Zhang, and S. Yu, "Enhancing crowd collaborations for software defined vehicular networks," *IEEE Commun. Mag.*, vol. 55, no. 8, pp. 80–86, Aug. 2017.

[33] X. Cheng, F. Lyu, W. Quan, C. Zhou, H. He, W. Shi, and X. Shen, "Space/aerial-assisted computing offloading for IoT applications: A learning-based approach," *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1117–1129, May 2019.

[34] T. Yang, Y. Hu, M. C. Gursoy, A. Schmeink, and R. Mathar, "Deep reinforcement learning based resource allocation in low latency edge computing networks," in *Proc. 15th Int. Symp. Wireless Commun. Syst. (ISWCS)*, Aug. 2018, pp. 1–5.

[35] F. D. Vita, D. Bruneo, A. Puliafito, G. Nardini, A. Virdis, and G. Stea, "A deep reinforcement learning approach for data migration in multi-access edge computing," in *Proc. ITU Kaleidoscope, Mach. Learn. 5G Future (ITU K)*, Nov. 2018, pp. 1–8.

[36] X. Chen, H. Zhang, C. Wu, S. Mao, Y. Ji, and M. Bennis, "Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4005–4018, Jun. 2019.

[37] J. Chen, S. Chen, Q. Wang, B. Cao, G. Feng, and J. Hu, "IRAF: A deep reinforcement learning approach for collaborative mobile edge computing IoT networks," *IEEE Internet Things J.*, vol. 6, no. 4, pp. 7011–7024, Aug. 2019.

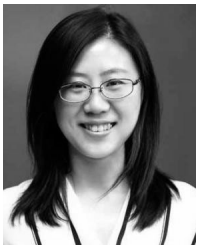
[38] X. Qiu, L. Liu, W. Chen, Z. Hong, and Z. Zheng, "Online deep reinforcement learning for computation offloading in blockchain-empowered mobile edge computing," *IEEE Trans. Veh. Technol.*, vol. 68, no. 8, pp. 8050–8062, Aug. 2019.

[39] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *Comput. Sci.*, vol. 8, no. 6, p. A187, 2015.

YUNZHAO LI received the B.S. degree from North China Electric Power University, Beijing, China, in 2018. He is currently pursuing the M.S. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His main research interest include edge computing.

FENG QI is currently a Professor with the Beijing University of Posts and Telecommunications, engaged in scientific research, teaching, and standardization research in information and communication. His research interests include communications software, network management, and business intelligence. He has won two National Science and Technology Progress Awards. He has also written more than ten ITU-T international standards and Industry Standards. He served as the Vice Chairman for ITU-T Study Group 4 and Study Group 12.

ZHILI WANG is currently an Associate Professor with the Beijing University of Posts and Telecommunications, engaged in scientific, technology, and standardization research work in communication networks and computer science. His main research directions are network management, communications software, and interface testing. He has won one National Science and Technology Progress Awards and wrote more than eight ITU-T international standards, and successively served as the Working Party Chair for ITU-T Study Group 2 and Working Party 2.



XIUMING YU received the master's degree in business administration from the University of International Business and Economics (UIBE), in 2013. She is currently an Engineer with the China Electronics Standardization Institute (CESI). She currently researches in the area of cyber-physical systems, smart manufacturing, and the industrial Internet.

SUJIE SHAO received the Ph.D. degree from the Beijing University of Posts and Telecommunication, Beijing, China, in 2015. He is currently a Lecturer with the Beijing University of Posts and Telecommunication. His research interests include edge computing, the Internet of Things, smart grids, and communication network management.