Received May 25, 2020, accepted June 22, 2020, date of publication June 25, 2020, date of current version July 7, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3004861

Semi-Online Computational Offloading by Dueling Deep-Q Network for User Behavior Prediction

SHINAN SONG[1], ZHIYI FANG[1], ZHANYANG ZHANG[2], (Member, IEEE), CHIN-LING CHEN[3,4,5], AND HONGYU SUN[6,7], (Member, IEEE)

[1]College of Computer Science and Technology, Jilin University, Changchun 130012, China
[2]Computer Science Department, The City University of New York, New York, NY 10017, USA
[3]Department of Computer Science and Information Engineering, Chaoyang University of Technology, Taichung 413, Taiwan
[4]School of Information Engineering, Changchun Sci-Tech University, Changchun 130022, China
[5]School of Computer and Information Engineering, Xiamen University of Technology, Xiamen 361024, China
[6]Department of Computer Science, Jilin Normal University, Jilin 136000, China
[7]State Key Laboratory of Numerical Simulation, Jilin 130022, China

Corresponding authors: Chin-Ling Chen (clc@mail.cyut.edu.tw) and Hongyu Sun (hilda.hongyu@gmail.com)

This work was supported in part by the Department of Science and Technology of Jilin Province under Grant 20190701002GH, and in part by the Cernet innovation Project under Grant NGII20180315.

ABSTRACT Task offloading could optimize computational resource utilization in edge computing environments. However, how to assign and offload tasks for different behavior users is an essential problem since the systems dynamic, intelligent application diversity, and user personality. With user behavior prediction, this paper proposes soCoM, a semi-online Computational Offloading Model. We explore the user behaviors in sophisticated action space by reinforcement learning for catching unknown environment information. With Dueling Deep-Q Network, both the prediction accuracy of users' behaviors and the server load balance are well-considered, while increasing the computational efficiency and decreasing the resource costing. We propose a dynamic simulation environment of edge computing to demonstrate that user behavior is the critical factor for impacting system utilization. As the action space increasing, Dueling DQN performs better than state-of-art DQN and other improved strategies, and also load balance in multiple different server scenario.

INDEX TERMS Edge computing, computational offloading, user behavior prediction, dueling deep-Q network.

I. INTRODUCTION

The recent advancement of Internet of Things (IoT) has motivated various applications with different requirements (e.g. Automatic Speech Recognition (ASR), Nature Language Processing (NLP), Computer Vision (CV) and accurate human-computer interaction (HCI)) [1]–[3]. Most of the applications require intensive computation resources to guarantee the Quality-of-Experience (QoE) and Quality-of-Service (QoS). At the same time, smart mobile devices have limited computation capabilities and battery powers. Therefore, computation offloading from edge users to servers becomes an essential part of mobile edge computing (MEC) systems to optimize resource utilization, energy consumption, and network delay [4].

There are variety of computation offloading methods such as [5]–[12] are proposed to offload task from edge users to servers in edge computing environments. However, most of the current computation offloading approaches assume that the edge users share the same settings. For example, they are homogeneous sensors or IoT devices, while edge users usually totally different in practical application scenarios since different algorithms they applied [11], [12]. Furthermore, human-driven intelligent applications would affect system performance by their personality behaviors [8].

Therefore, considering the differentiation between user equipment (UE) to optimize the performance of computation offloading further is an essential issue in MEC system. But the diversity of edge users brings at least the following

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

TABLE 1. Models and algorithms of state-of-the-art works.

Reference	Key differences in system model				Offloading strategy	Action space size	Algorithms
	Sever	Energy	Channel	User			
[7]		✓			Offline	2	Convex optimization
[8]	✓				Offline	2	Control theoretic
[9]		✓			Online	2	Game theory
[14]	✓			✓	Semi-online	2	Queue theory
[10]	✓				Offline	7	DQN
[15]			✓		Online	16	DQN
[16]	✓		✓		Online	17	DQN
[17]	✓	✓	✓		Semi-online	100	DNN+Monte Carlo Tree Search+RL
[18]	✓				Semi-online	150	DQN+Adaptive Genetic Algorithm
[19]			✓		Semi-online	15	Q learning+ Monte Carlo Tree Search
[20]	✓	✓	✓		Semi-online	2	DQN+Double DQN
[21]		✓	✓		Semi-online	60	DNN+ Convex optimization +RL
[22]	✓				Semi-online	15	DNN+ Convex optimization +RL
[23]			✓		Semi-online	30	Q learning+DNN+RL

challenges in computation offloading procedure: i) the state space of the UE set would increase exponentially as the growth of UE number (shown in FIGURE 1); ii) the UE behavior observation is more difficult for server part since the diversity between edge users. To offload the tasks between different UE and servers with high performance, in this paper, we introduce semi-online architectures and Dueling Deep-Q Network (Dueling DQN) [13] to optimize resource utilization, energy consumption, and network latency in computation offloading procedures.

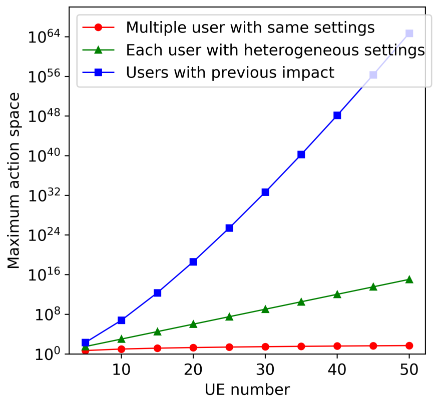


FIGURE 1. Action space trend as UE number increasing.

To conclude, the main contributions of this paper are as follows:

1) We propose *soCoM*, which offloads tasks reasonably by predicting UE behavior differences without further communications by guidelines or labeled data. soCoM is a semi-online distributed offloading model for an edge computing system. soCoM utilizes Markov Decision Process and reinforcement learning to make decisions and explore the unknown dynamic user information. Compared to state-of-art works, soCoM is more suitable for a diversity system with better computational efficiency.
2) As far as we know, this is the first time to use Dueling Deep-Q Network to explore the large UE action space to optimize task scheduling procedures. We investigated three popular variants of DQN and analyzed and verified their characteristics in massive action space exploration. Dueling Deep-Q Network could optimize two aims in ample action space: user behavior prediction and server load balance.
3) We propose a dynamic simulated environment for edge computing containing different users and servers. Experimental results demonstrate that user behavior prediction is the critical factor for system resource utilization. As the action space increasing, Dueling DQN performs better than state-of-art DQN and other improved strategies. For different multiple servers scenario, UE behavior prediction guides the two-way choice between UE and server. Dueling DQN balances the load among servers with more benefits.

The rest of this paper is organized as follows. Section II introduces related work about RL and computational offloading. Section III describes the semi-online edge computing system model. Section IV gives the model soCoM, which consists of behavior prediction with DQN-based algorithms. Section V presents the improvement of soCoM with Dueling DQN. The simulation results are in Section VI. Finally, Section VII summarizes our work.

II. RELATED WORK

Resource optimization and scheduling are essential works for computational offloading in edge computing environments. The design goals for computational offloading are to optimize resource utilization, energy consumption and network delay. Current works in TABLE 1 show that RL-based method could optimize the performances when multiple users share the same parameters, while our work is to resolve the computational offloading problem when the users have different settings in practical applications.

The approach proposed in article [7] decomposes the offloading process into several sub-problems to resolve the NP-hard problem in computational offloading procedures. The method proposed in paper [8] uses Control-Theoretic further to decompose the sets of the sub-problems for performance optimization. The approaches proposed in article [9] and [14] introduces users' behaviors to guide the computational offloading procedures, in which [14] proposed a heuristic method to resolve the NP-hard problems. Compared with these works, *soCoM* considers the differences between multiple-users, and try to optimize the resource utilization between servers and different end systems by exploring Dueling Deep-Q Network (Dueling DQN) under semi-online architectures.

Current approaches such as [10], [15], [16], [20], [22] also uses reinforcement learning (RL) methods to resolve the computational offloading problems in edge computing environments. Reference [10] uses Deep-Q Network (DQN) and job shop scheduling problems for computational offloading. Reference [15] assume that the servers have the knowledge of the connected channels while do not know the knowledge of the offline channels. Reference [16] proposes Q-learning based and Deep Reinforcement Learning (DRL) based schemes for a multi-user mobile edge computing (MEC) system. Reference [22] explores DQN to handle the offloading request and resource allocation in each sub-problem decomposed in computational offloading procedures. Reference [20] uses multi-agent reinforcement learning to meet the requirements of individual user such as resource allocation, channel states, and quality of service (QoS). As the typical RL method, Deep Reinforcement Learning (DQN) has some limitations when the action space is ample. It is challenging to get excellent training for complex scheduling in the dynamic edge computing environment [18]. Generally, to achieve better results, other heuristic algorithms and pre-trained neural networks are needed. In the work of Chen *et al.*, the DQN-based method should learn a policy by experience transitions with a well-trained Deep Neural Network (DNN) [17]. In order to optimize computational offloading under our settings with high performances, we apply Dueling DQN, an improved DQN method for resource resign and allocation in MEC system.

Current approaches such as [21], [23] have other assumptions that the offloading procedures are independent processes without continuous impact between users and servers. To solve changeable MEC conditions, the approach proposed in article [24] uses directed acyclic graph (DAG) to model the dependent tasks. It explores a two-layered reinforcement learning (TLRL) algorithm for resource-constrained UE in MEC, which optimizes the utilization efficiency and offloading latency simultaneously by introducing a weighted reward. Compared with the current work, we propose a semi-online model to consider the impact presented by the relationships between users and servers.

III. SYSTEM MODEL

The design goals of *SoCoM* are to optimize the resource allocation and energy consumption of UE. To better define the problem and potential resolutions, we explore a distributed scheduling strategy for a semi-online system with binary offloading, which contains three sub-models: i) a computing model, ii) a communication model, and iii) an energy consumption model, as shown in FIGURE 2. The symbols and descriptions used in the remainder of this paper are listed in TABLE 2.

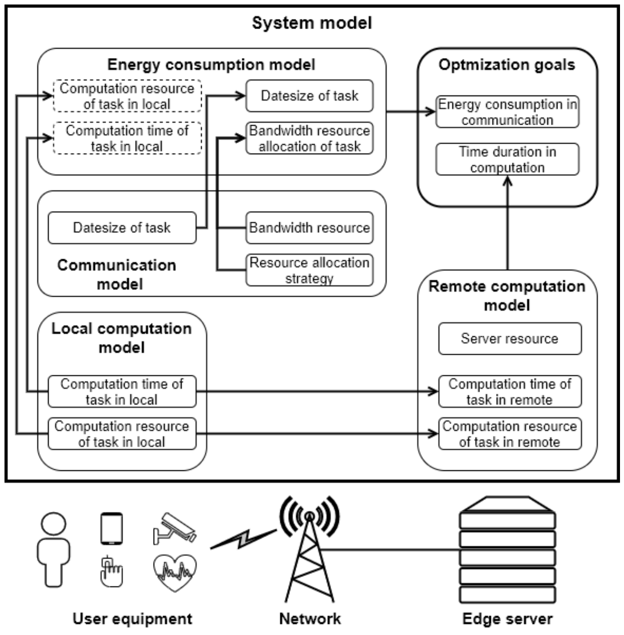

System model
Energy consumption model
Computation resource of task in local
Computation time of task in local
Datesize of task
Bandwidth resource allocation of task
Optmization goals
Energy consumption in communication
Time duration in computation
Datesize of task
Bandwidth resource
Resource allocation strategy
Communication model
Remote computation model
Server resource
Local computation model
Computation time of task in local
Computation resource of task in local
Computation time of task in remote
Computation resource of task in remote
User equipment
Network
Edge server

FIGURE 2. Edge computing system model with UE, network and edge server.

The basic structure of *soCoM* consists of one edge server and several edge users. The UE set is $u^{(k)} \in U = \{u^{(1)}, u^{(2)}, \dots, u^{(n)}\}$. For each UE $u^{(k)}$, there is a task data set $D^{(k)} = \{d_1^{(k)}, d_2^{(k)}, \dots\}$. In which, $d_i^{(k)}$ is the task generated by the user equipment at time i. The arrival of each task corresponds to a new task $t_i^{(k)}$, and the resource allocation rate of the task is $\theta_i^{(k)}$. The total communication bandwidth provided by the system is C_{max}, and the total number of resources (such as the number of CPU cores or virtual machines) is E_{max}.

A. COMPUTATION MODEL

Computation model plays an important role in system performance optimization in traditional cloud computing or distributed computing systems. For example, the execution time of programs with different instructions would affect the energy consumption of CPUs. However, traditional computation model does not work well in edge computing since the three main reasons: i) UEs are heterogeneous in practical edge computing scenarios, ii) the configures of equipment used by each edge user are unseen to server-side, and iii) the algorithms (e.g. CNN/DQN/RNN) used for tasks also impact the diversity of execution efficiency [25]. Therefore,

TABLE 2. Symbol description.

Symbol	Description
$u^{(k)}$	User equipment k
$t_i^{(k)}$	i-th task of user equipment k
$d_i^{(k)}$	i-th task data of user equipment k
$\Pi_{k,i}^L$	Task local computation model
$\Pi_{k,i}^R(T)$	Task remote computation model
$\Pi_{k,i}^C(T)$	Task communication model
$\Pi_{k,i}^E(T)$	Task energy consumption model
π^R	Server computation ability allocation policy
π^C	Server bandwidth allocation policy
π^E	Server energy consumption policy
C_{max}	Total system communication bandwidth
$C_{left}(T)$	System communication bandwidth left
E_{max}	Total system computing resources
$E_{left}(T)$	System computing resources left
$\alpha_i^{(k)}$	Task local execution time
$\alpha_{k,i}^{(R)}$	Task remote execution time
$\beta_i^{(k)}$	Task transfer time
$\gamma_i^{(k)}$	Task local resource occupancy
$\gamma_{k,i}^{(R)}$	Task remote resource occupancy
$c_i^{(k)}$	Task transmission takes up bandwidth
$\epsilon_i^{(k)}$	Task transmission energy consumption
$s_i^{(k)}(T)$	The status of the task at time T

we propose a time-based computation model to achieve better abstraction in *soCoM* system.

In the computation model, the attributes of each task include the generators, the generation time, the generated frequency, the type of generated data, the amount of the generated data, application type, the local execution time, and the flag whether the task has been offloaded. Our offloading model is a binary offloading [26]. Part of the necessary information would be sent to the edge server for decision making when the task is to be offloaded. Besides, this essential information is also the criteria for prioritizing the task execution of the user equipment, such as using Short Job First (SJF) or First Come First Served (FCFS) strategy.

Computation models are consist of local computation models and remote computation models, which measure the resource consumption of tasks during local or remote computing. Let the i^th task of user k be $t_i^{(k)}$. The task's local computation model is shown as Equation 1.

$$\Pi_{k,i}^L = [\alpha_i^{(k)}, \gamma_i^{(k)}] \tag{1}$$

When the remote server offloads a task, the computing state of the task is related to the server's computing resources. The change of the server's remaining computing resource $E_{left}(T)$ is a function of time T. Then the task's remote computing model $\Pi_{k,i}^R(T)$ is shown as Equation 2.

$$\Pi_{k,i}^R(T) = [\alpha_{k,i}^{(R)}, \gamma_{k,i}^{(R)}, E_{left}(T)] \tag{2}$$

where, $\alpha_i^{(k)}$ is the local computing time of the task, $\gamma_i^{(k)}$ is the local computing resource allocation rate of the task. $\alpha_{k,i}^{(R)}$ is the remote computing time of the task, and $\gamma_{k,i}^{(R)}$ is the remote computing resource allocation of the task.

B. COMMUNICATION MODEL

The communication model $\Pi_{k,i}^C(T)$ is consist of the total bandwidth resource C_{max} provided by the server, the remaining bandwidth resource $C_{left}(T)$, the bandwidth resource allocation strategy π^C, and the amount of transmitted data $d_i^{(k)}$, the model is shown as Equation 3.

$$\Pi_{k,i}^C(T) = [C_{max}, C_{left}(T), \pi^C, d_i^{(k)}] \tag{3}$$

In which the bandwidth allocation function π^C determines the number of channel bandwidth could assign to specific task during computation offloading process.

In our simulation system, we improve the communication model based on paper [27]. In their work, multiple edge users share the same channel for computation offloading has limitations. Unless one user has released the channel, another user could not allocate the channel. Therefore, we define our communication model for advancing performance optimization. The remaining bandwidth of the current system is $\omega = C_{left}(T)$. The UE set in this round for offloading decision making is $U_{off}(T)$. After offloading, the bandwidth usage obtained by each user during the offloading procedure is calculated by Equation 4.

$$c_i^{(k)} = \omega \log_2(1 + \frac{q^{(k)} g^{(k,s)}}{\omega_0 + \sum_{u^{(i)} \in U_{off}(T)} q^{(i)} g^{(i,s)}}) \tag{4}$$

where $q^{(k)}$ is the energy transmitted by the wireless network base station between the user equipment and the edge server. ω_0 is the power of background noise. This model has the following characteristics: users would interfere each other significantly when many users choose to offload task at the same time, and the transmission rate would reduce sharply since the interference. Therefore, the resource allocation introduce challenges as choosing combinations of equipment of edge users. we assume $d_i^{(k)}$ is the upstream data volume, the relationship between data volume and transmission time is calculated by Equation 5.

$$\beta_i^{(k)} = \frac{d_i^{(k)}}{c_i^{(k)}} \tag{5}$$

C. ENERGY CONSUMPTION MODEL

The limited power of mobile devices is an essential obstacle to handle big volume data in computing tasks. Therefore, how to reduce the energy consumption in MEC system has significant values in practical applications. The energy consumption model applied in our system contains two parts: local computation energy consumption sub-model and transmission energy consumption sub-model between edge users and server. The local computing energy consumption is stable with the full load, which is generated by the UE continuously in the computation offloading procedures. Besides, the time-based abstraction of UE computing amount also applies to those UE without regional computing ability.

Therefore, we mainly design the communication energy consumption in this paper, as shown in Equation 6, where $d_i^{(k)}$ is data amount and $\beta_i^{(k)}$ is data transmission duration. π^E is the energy model for a specific application environment.

$$\Pi_{k,i}^E(T) = [d_i^{(k)}, \beta_i^{(k)}, \pi^E] \tag{6}$$

In our simulation, we choose the energy model used in article [28]. The energy consumption of the data transmission process based on WiFi or Long Term Evolution (LTE) wireless networks is the relationship between up-link throughput t_u (Mbps) and downlink throughput t_d (Mbps). The appropriate power level W for each equipment of edge users could be calculated by Equation 7.

$$W = \theta_u t_u + \theta_d t_d + \theta_i \tag{7}$$

According to the research in article [28], the recommended model parameters for WiFi networks are $\theta_u = 283.17, \theta_d = 137.01, \theta_i = 132.86$, and for LTE networks is $\theta_u = 438.39, \theta_d = 51.97, \theta_i = 1288.04$.

In addition, $\epsilon_i^{(k)}$ is the transmission energy consumption during task offloading. $w_i^{(k)}$ is the average transmission power level required for the task offloading in one time. The relationship between transmission energy consumption and communication time calculated by Equation 8.

$$\epsilon_i^{(k)} = w_i^{(k)} \beta_{i,j}^{(k)} \tag{8}$$

D. OPTIMIZATION GOALS

We represent the optimization goals of the system based on cost-benefit analysis. In the computation offloading process, energy consumption of data transmission is the cost of UE, while the computing resource obtained from the server is the benefit of UE. We formulate this problem to optimize the resource utilization and energy consumption of UE under the constraints of total server resources. For multi-user MEC system, we take the above two optimization objectives for all continuous offloading decisions, which is shown in Equation 9.

$$\begin{aligned} OPT^* &= \max_{\pi \in \Pi} \sum \frac{\alpha_i^{(k)}}{\epsilon_i^{(k)}} \\ s.t. &\sum c_j^{(k)} < C_{max}, \quad s_j^{(k)}(T) = 0, \ \forall T \\ &\sum \gamma_{k,l}^{(R)} < E_{max}, \quad s_l^{(k)}(T) = 1, \ \forall T \end{aligned} \tag{9}$$

where $s_i^{(k)}(T)$ is the state of the task at time T. Tasks change with user behaviors and system time. When $s_i^{(k)}(T) = 0$, the state of the user is "transmitting", when $s_i^{(k)}(T) = 1$, the state of the task is "remote waiting", when $s_i^{(k)}(T) = 2$ the state of the task is "executing remotely", and when $s_i^{(k)}(T) = 3$, the state of the task is "completed remotely".

The optimization goal is to maximize the number of computing resources obtained while minimizing energy consumption per unit in all possible offloading strategies. π is one offloading policy selected by the system, and Π is the set of all reasonable offloading policies. Besides, the optimization goal also needs to meet two resource constraints: computing resource constraints and communication resource constraints. They indicate that the computing resources allocated by the server cannot exceed the upper limit E_{max} provided by the server at any time. At the same time, the communication resources allocated in the transmission channel must not exceed the upper limit of the bandwidth C_{max} provided by the system.

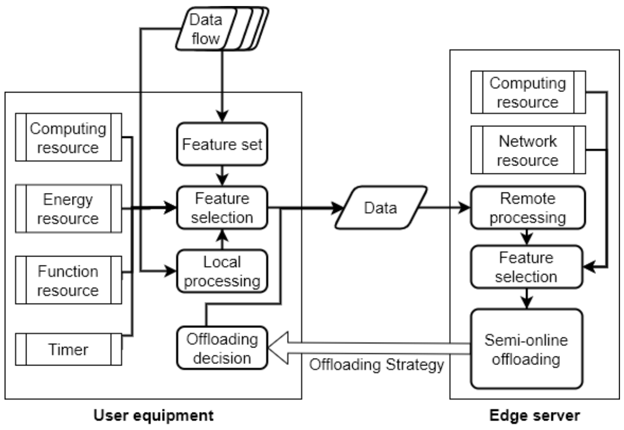


FIGURE 3. Semi-online computational offloading process.

IV. SEMI-ONLINE COMPUTATIONAL OFFLOADING MODEL (soCoM)

Server needs to select appropriated UE for computation offloading based on dynamic computing parameters includes quality of transmission channel, the queue in server side, and diversity user behaviors. The design goals are to optimize the resource utilization and energy consumption by using cost-benefit model. Especially for multi-user systems, the computational offloading in the edge environment is a complex continuous action decision. One offloading would affect the resource cost for all tasks within the duration. Reinforcement learning has a significant advantage in solving such problems. In this chapter, we will propose the semi-online Computational Offloading Model (soCoM) based on reinforcement learning.

A. MARKOV DECISION MODEL

We use Markov Decision Process to explain the decision process for computation offloading. Markov decision process is described by a five-tuple (S, A, P, R, γ). $a_i = \{Z\} \in A$. It is a limited set of *actions* to decide for offloading. The set element is a numeric tuple from 1 to 2^n, including the UE ID that needs to offload at the i-th decision. When the system makes an offloading decision, first it select a value from the action set. We convert the value to binary, and the number 0 is filling in n digits from the high order to obtain a 0-1 sequence of length n. Each bit in the sequence corresponds to one UE, which is denoted as $a_i^{(k)}$. When $a_i^{(k)} = 0$, UE k executes locally. When $a_i^{(k)} = 1$, UE k offloads the task to the edge server.

R is a *reward* function, which refers to the reward immediately obtained from the environment after an action is complete. For example, when to encourage users to offload, the offloading action would get a positive immediate return. For the entire system or the ultimate optimization goal, we need to calculate the cumulative reward after performing a series of decision actions.

Then, we could get a trajectory sequence $\tau = \{s_0, a_0, s_1, a_1, \ldots\}$ containing *states* and *actions*. The purpose of reinforcement learning is to find the optimal strategy π, which can maximize the cumulative return expectation of τ. In this paper, we need to find the optimal offloading strategy, which is described by a series of offloading decisions to maximize the expectation of calculation amount under the unit transmission energy consumption, which is shown in Equation 9.

B. REWARD DELAY

System state is changing over time, which is shown in Figure 4. Assume at the moment of T_k, the core decision algorithm DQN obtains the observation $Observation_k$ from the server state and offloads the action $Action_k$ according to the observation. The offloading action would affect the states of UE who received the offloading message, and then the specific task states turns to offload. After the task enters the offloading process, it would undergo state transitions such as transmission, reaching the server cache, server processing, and execution completion. During the entire conversion process, server resources and status would be continuously affected. When the task is completed, the reward $Reward_{k+1}$ obtained from this offloading returns to the decision algorithm for learning. From the description, at the time of T_k, the decision algorithm DQN could not obtain the immediate reward of this action, but only the reward of the previous moment. Therefore, continuous decision making in reinforcement learning processes is critical.

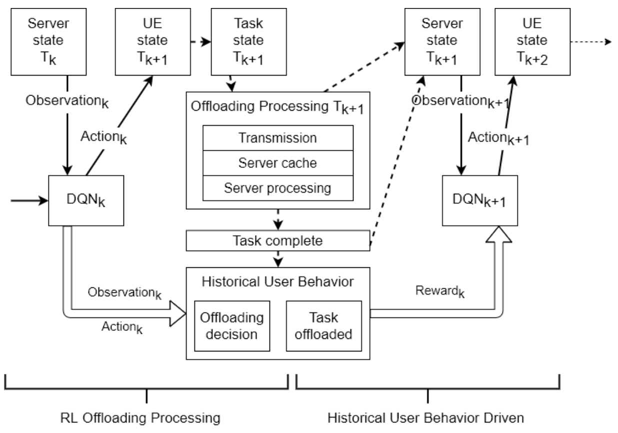


FIGURE 4. soCoM with DQN based on user behavior prediction.

The system *states* set is a 5-tuple: S, including system resource allocation rate, channel allocation rate, the number of tasks that have been transmitted but not reached, the number of tasks that have been executed but not completed, and the remaining execution time of tasks. $s_i^{(k)}(t) \in \{LW, RT, RW, RP, COM\}$ is the state of task i at time t, including local waiting (LW), remote transmission (RT), remote waiting (RW), remote processing (RP), and completed (COM).

The set of *actions* includes the set of all edge equipment to be selected. Action set $A = [a^k], k = 1, 2, \ldots, N, a^k \in 0, 1$. Reinforcement learning is rewarded by the return function R_j at the time of t. For a semi-online system environment, the remote server could only obtain task information that has offloaded to the remote. Therefore, we consider the saved running time with energy consumption of the task as a reward for an offload behavior at time t, which is shown in Equation 10.

$$\begin{aligned} Reward_t &= \frac{\alpha_i^{(k)}}{\epsilon_i^{(k)}} \\ s.t.\ s_i^{(k)}(t) &= COM, \quad \forall k, i \\ s_i^{(k)}(t-1) &= RP, \quad \forall k, i \end{aligned} \tag{10}$$

Besides, the system would receive a punitive reward when resources are overloaded. We set a punitive reward as the negative value of the absolute value for the current system reward. At the same time, it could ensure that the intensity of punishment is consistent with the degree of reward obtained during operation. The punitive reward is shown in Equation 11:

$$Reward_t^- = -|Reward_{t-1}| \tag{11}$$

The cumulative reward of reinforcement learning is the lower limit of the system's optimization goal OPT^*, which is shown in Equation 12.

$$\sum_{t=T}(Reward_t + Reward_t^-) \leq OPT^* \tag{12}$$

A model-free reinforcement learning method: the time difference method (TD) is applied to solve the optimization problem. The time difference method in reinforcement learning combines Monte Carlo sampling and bootstrapping in dynamic programming methods, which could bring better learning results and efficiency. For specific reinforcement learning methods, due to the successful application of channel selection calculation offloading [15], we use Deep Q Learning to solve this problem. In the structure of neural network, we use the L2 Mean Squared Error (MSE) loss function [29], which is shown in Equation 13.

$$MSE(y, y') = \frac{\sum_{i=1}^{n}(y_i - y_i')^2}{n} \tag{13}$$

The semi-online reinforcement learning algorithm running on the remote server is in Algorithm 1. The *GetReward*() method obtains the instant system reward.

Algorithm 1 Semi-Online Computational Offloading Model With DQN

```
1: At the beginning of time 0: Initialize $Q(s, a), \forall s \in S, \forall a \in A(s), \gamma$
2: for episode in 1,2,...,n do
3:     Get $s$ from environment
4:     Begin remote server processing:
5:     while True do
6:         Get $a_j$ from $A(s_j)$
7:         Get $a_{(j)}$ from $A(s_{(j)})$
8:         Refresh $Q(s_{(j)}, a_{(j)})$
9:         $R_{(j)} = GetReward()$
10:        Offloading tasks from UE in $a_{(j)}$
11:        $s_{(j)} = s_{(j+1)}, a_{(j)} = a_{(j+1)}$
12:        if task $t_i^{(k)}$ is finished: then
13:            $reward(t_i^{(k)}) = \frac{\alpha_i^{(k)}}{\epsilon_i^{(k)}}$
14:        end if
15:        $R_{(j+1)} = GetReward()$
16:        Training processing
17:    end while
18: end for
```

V. IMPROVING soCoM WITH DUELING DQN

Our motivation is that, because of behavioral differences, UE selection leads to large action space. First, we discuss how to take advantage of user behavior prediction for a complex multiple server system. Then, we explore the principles of several improved methods based on DQN and give reasons for choosing Dueling DQN as the target algorithm.

We divide our optimization goal, OPT^*, into two sub-goals:

1) *Improve choices probability for well-behavior UE.* In the case of the same throughput, the more well-behavior UE the system select, the higher profit it would gain.
2) *Improve offloading throughput avoiding resource bottleneck.* With equal rewards for single offloading, the more offloading the UE complete, the more benefit it would get.

Reinforcement learning starts with random exploration. When the action space is enormous, covering all situations is challenging. We hope that RL methods could consider both of sub-goals. However, bootstrapping in DQN explores each action equally. When the action space is too large, DQN wastes unnecessary energy on studying those actions with lower rewards. Smart RL methods could obtain approximately optimal decisions through limited exploration.

A. COMPLEX AND LARGE SCALE ACTION SPACE

In this article, the large-scale action space is not equivalent to large-scale edge computing systems. Indeed, large-scale system scheduling involving thousands of UE and dozens of servers is a challenging problem. However, there is usually no noticeable difference in behavior between those UE in large-scale systems.

Let us illustrate this problem with examples. In the issue of channel selection, we assume that there are m channels with different conditions and n UE with the same behaviors. Each offload selection only needs to select one of the m channels. The possible offloading action space is $C_n^m \times m!$.

But suppose there are n UE with completely different behaviors. During the process of allowing several UE to offload each round, the possible offloading action combinations is shown in Equation 14.

$$\sum_{i=0}^{n} C_n^i = 2^n \tag{14}$$

If the order of offloading UE in each round would affect the resources allocation for others, for example, in FCFS, the UE task who arrives first would occupy the resource first. The possible offloading action space may be $\sum_{i=0}^{n} A_n^i$.

Besides, the number of UE is usually much larger than the number of channels. Therefore, the behavior difference between users would significantly increase the scheduled action space in the learning process, which makes optimization difficult.

B. TWO-WAY CHOICE BASED ON BEHAVIOR PREDICTION

When the edge system consists of multiple UE and servers, UE could choose one of the servers to obtain services, and the server would accept various offloading requests at the same time. Therefore, there is a two-way choice problem between UE and servers: i) UE can choose a server that provide services better and faster, and ii) when resources are limited, servers choose to serve which UE first.

We define a two-way priority descriptor $p_{k,j}(t)$ between UE k and server j, which describes the priority of UE k and server j to establish the offload service at time t.

Therefore, when selecting a server, the UE choose preferentially those servers with a high probability of QoS according to $p_{k,j}(t)$. When the server selects UE, it would give priority to UE with better resource utilization. According to the analysis, the two-way priority descriptor $p_{k,j}(t)$ equals to the behavior prediction of UE.

The multiple server system with soCoM is in FIGURE 5. The core idea of DQN is guiding the *action* through *reward* functions based on observation of system *state*. In soCoM, we use *reward* as the UE's behavior prediction, which brings two benefits: i) The reward value measures the pros and cons of UE behavior and assists the server and UE in making a two-way choice. ii) As the RL algorithm gradually learning the right actions and corresponding rewards during the training process, the performance of the entire system could improve.

C. DOUBLE DQN

We first focus on Double DQN [30]. Compared with DQN, Double DQN uses an empirical playback strategy to solve

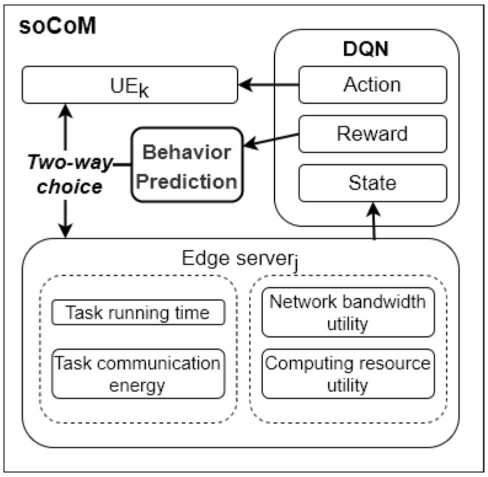


FIGURE 5. soCoM for multiple servers system.

the problem of DQN overestimation, which reflects in the different goals of TD in Equation 15 and 16.

$$Y_t^Q = R_{t+1} + \gamma max_a Q(S_{t+1}, a; \theta_t) \tag{15}$$

$$Y_t^{Double} = R_{t+1} + \gamma Q(S_{t+1}, argmax_a Q(S_{t+1}, a; \theta_t^-); \theta_t) \tag{16}$$

Double DQN improves DQN's decision-making effect on a large action space to a certain extent. Because of the over-fitting problem, DQN could not estimate every point of the value function during the training process. That is, the estimator is not uniform. The overestimation of DQN median function affects the final decision. Therefore, Double DQN selects and evaluates actions through different value functions, which reduces the over-fitting problem of DQN. Since reinforcement learning is a process of learning while exploring, good action choices would lead to better exploration paths.

D. PRIORITIZED REPLAY

Prioritized Replay optimizes DQN for improving the learning efficiency of DQN [31]. By enhancing the max operation in DQN, Double DQN still adopts a uniform distribution during empirical playback. The Prioritized Replay strategy considers that, for agents, not all historical data is equally meaningful for learning. By breaking the uniform sampling and assigning sampling weight to the state, Prioritized Replay improves the learning efficiency.

The principle of Prioritized Replay is by sorting δ of TD-error. The larger the TD-error, the more significant the gap between the state value function and the target. The more considerable the update amount of the agent, the higher the learning efficiency. We let the TD-error at sample i be δ_i, and the ranking result of all δ_i is $rank(i)$, then the importance sampling coefficient at this place is shown in Equation 17.

$$p_i = \frac{1}{rank(i)} \tag{17}$$

$$P(i) = \frac{p_i^\alpha}{\sum_k p_k^\alpha} \tag{18}$$

$$\omega_i = (\frac{1}{N}\frac{1}{P(i)}) \tag{19}$$

However, Prioritized Replay does not solve the problem of insufficient exploration as the action space increases. Sorting historical data would cause the agent to concentrate actions in a smaller range, which is not conducive for exploring new action sequences.

E. DUELING DQN

Dueling DQN considers that not all actions are essential [13]. Especially when some individual states appear, some key actions are beneficial. Dueling DQN focuses on the relationship between critical states and actions. Dueling DQN would solve the problem of too much action space caused by UE differences.

Dueling DQN changes the structure of DQN network, as shown in Figure 6. The top one is basic DQN model, and Dueling DQN is structurally consistent with the original DQN in the first half. At the output, Dueling DQN does not directly connect a fully connected layer after the convolutional layer but maps the output to two fully connected layers. These two fully connected layers would evaluate the value and advantages of actions and states, respectively. They are also called value function $V^\pi(s)$ and potential function $A^\pi(s, a)$. Finally, Dueling DQN obtains the final Q value output by merging the two fully connected layers, which is shown in Equation 20.

$$Q^\pi(s, a) = V^\pi(s) + A^\pi(s, a) \tag{20}$$

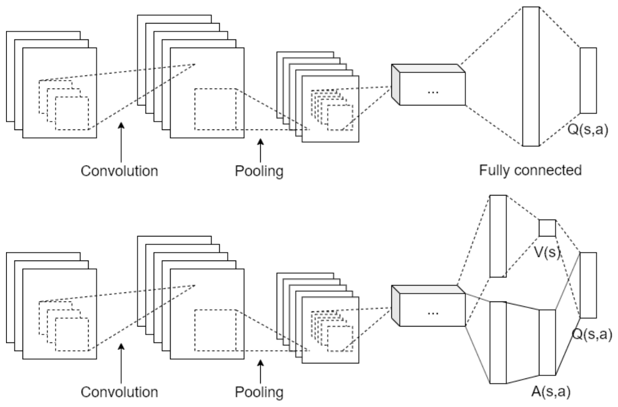


FIGURE 6. Deep Q network(top) and Dueling Deep Q network(bottom).

Expanding the above formula we can get Equation 21.:

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + A(s, a; a\theta, \alpha) \tag{21}$$

Among them, θ, α, β are neural network parameters. According to analysis, $Q(s, a; \theta, \alpha, \beta)$ is the true Q function estimate. So the value functions $V(s; \theta, \beta)$ and potential functions $A(s, a; \theta, \alpha)$ do not provide an accurate estimate of the value of the action. To solve this problem, Dueling

DQN modified the potential function to forcefully reduce the proportion of the potential function in action selection, which is shown in Equation 22.

$$Q(s, a; \theta, \alpha, \beta) = V(s; \theta, \beta) + (A(s, a; \theta, \alpha) - \frac{1}{|A|} \sum_{a'} A(s, a'; \theta, \alpha)) \tag{22}$$

VI. SIMULATION RESULT

In this section, we demonstrate the performance of soCoM through simulation. In the following simulations, we use time units to simulate the amount of calculations performed by different UE and servers. We use TensorFlow 1.0 to implement our RL algorithm in Python and use SimPy to complete the simulation experiment [32]. The configuration of the experiment is in TABLE 3.

TABLE 3. Single server experiment parameter.

Description	Value
Basic Transmitt Datasize	64KB
Total Bandwidth	50Mbps
Basic Running Time per Task	25ms
Basic Local Resource Obtain per Task	10%
Number of User Equipment	5-20
Computation Rate	4

The tasks generated by applications have randomness in the number of instructions and the size of the input/output files. Based on the real measurement data offloaded from the image recognition calculation in [33], we generate simulation data by adding uniform noise.

In the single-server simulation experiment, we set up 5 to 20 user equipment and one edge server. The minimum computing time unit is 25ms. Different hardware platforms, algorithm platforms, and input data sizes result in different task execution times. Changes in the system environment (such as user-side or network congestion) would affect task execution and communication time. In the simulations, as UE ID k increases, the amount of calculations for user tasks is increasing. ξ is random noise from 1 to 5:

$$\alpha_i^{(k)} = (k + 1) \times 25 \times \xi \tag{23}$$

The minimum data volume unit is 64KB. As UE ID k increases, the amount of transmission data size is decreasing, which simulates user status with different tasks and data volumes. n is the total number of UE:

$$d_i^{(k)} = (n - k) \times 64 \tag{24}$$

A communication channel connects UE and the server. The total network bandwidth in the non-blocking state is 50 Mbps. The offloading algorithm would determine which UE can get the offloading opportunity in this round, and the requesting user could complete the offloading task selection based on the message. The simulation time is 150s. In the process, when UE has successfully offloaded, a new task would be generated to ensure the continuous generation of the task flow.

For experiments with non-RL methods, we first select an online algorithm, a semi-online algorithm, and an offline algorithm:

1) *Online algorithm.* The resource selection process initiated by the user terminal is online scheduling. Users need to complete decision optimization through online information exchange [27]. In our simulation, the online algorithm could not obtain any UE information, so the server selects the target UE set for offloading randomly.
2) *Semi-online algorithm.* SPaC is a semi-online algorithm based on communication time [14]. In the decision-making process, SPaC makes the offloading decision by selecting the UE combined with the shortest communication time.
3) *Offline algorithm.* In principle, the offline algorithm could obtain an optimal global solution. Due to such complex scheduling is NP-hard [7], we could not get a continuous optimal offline schedule within a productive calculation time. In our simulation, the offline algorithm assumes that the calculation amount of the task and the transmission data size are known. In each one round of decision-making, the offline algorithm selects the UE set that maximizes the optimization target only in one round and does not consider the continual impact.

For experiments with RL, we select the basic DQN algorithm, Dueling DQN, Double DQN algorithm, and Prioritized Replay. For the neural network implementation, we use a fully connected neural network (DNN) consisting of one input layer, two hidden layers, and one output layer. The number of neurons in the hidden layer is 20. Adam's optimized learning rate is 0.01, the training batch size is 32, and the memory size is 512. The source code is available at https://github.com/snsong/soCoM.

We analyze the algorithm's adaptation to different action spaces and its response to changes in the dynamic system environment. This experimental design has the following considerations:

1) *UE cost and benefit:* Each UE has different behaviors. As the UE number grows, the spatial range of algorithm actions increases exponentially. The experimental indicators include offloading throughput, optimization goals, saved computing time, and communication energy consumption. Here we have not considered the user's local energy consumption of calculation. As UE is heterogeneous, UE computing energy saving is usually proportional to the amount of calculation and time required to complete the offload.
2) *Server load and profit:* When the server resources are constant, the UE number increasing would bring challenges to the server load. The flexible scheduling could balance system resources to avoid congestion and idleness. Besides, in practical applications, the server

obtains profits by providing computing resources. By comparing the utilization of network bandwidth, computing resources, and optimization goals, it is possible to measure the benefit received by the server from an offload algorithm.

3) *UE behavior prediction:* Furthermore, what makes the algorithm give proper scheduling under limited resources? We analyzed the selection of UE during the execution of different RL algorithms. The result explains that the prediction of user behavior would affect the efficiency of offloading.

A. PARAMETER ANALYSIS

We first focus on two essential parameters in RL: e-greedy and reward delay, see in FIGURE 7. We use e-greedy method for reinforcement learning exploration [34]. The parameter ϵ determines when the decision is exploring. The behavior that e-greedy chooses to execute each time is the most considerable estimated value. In the case of a small probability, the algorithm selects other actions randomly. For example, when $\epsilon = 0.1$, RL would choose the given solution with a probability of 0.9 and conduct a random exploration with a probability of 0.1.

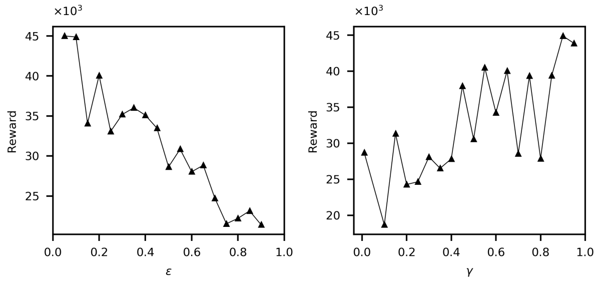


FIGURE 7. **RL parameters of e-greedy(left) and reward delay(right).**

Reward decay is the parameter γ in Equation 15 and 16. Reinforcement learning depends on the reward hypothesis, and all goals are to maximize the estimated cumulative rewards. However, the probability of getting rewards decreases as the algorithm goes further. Because of the difficulty increases, the discount rate γ increase, either, which is between 0 and 1. The larger the γ value, the less the discount, the learning agent pays more attention to the long-term reward. If γ is smaller, the agent pays more attention to the short-term reward.

We examined the setting of the two key parameters from the perspective of model optimization reward with a penalty, see in Equation 11 and 12. According to the results, when the value of ϵ is small, a better learning effect can be obtained. And γ needs to be larger to obtain a better learning effect. Therefore, soCoM pays more attention to the impact of short-term benefits on the system. In the following experiments, the values of these two parameters are: $\epsilon = 0.1$ and $\gamma = 0.9$.

B. UE COST AND BENEFIT

We compare the experimental results obtained by the non-RL methods and the RL methods in simulation. From the experimental results, the RL-based methods achieve good preliminary results in most cases. The offline algorithm and SPaC is an optimal solution based on the one-round decision, but not the globally optimal solution for continuous actions. As the action space increasing, the single optimal of the offline algorithm gradually leads to a better global solution. But considering the computational complexity, it is not suitable for the real world.

From the analysis of the optimization goal OPT, when the number of UE is increasing, Dueling DQN still maintains good results, see in FIGURE 8. Moreover, Dueling DQN adopts to select UE with lower transmission energy consumption and a higher calculation amount to achieve the optimization goal. In the same simulation for 10 UE, the energy consumption of Dueling DQN is 58.1% and 32.1% of DQN and Double DQN, and saved computation capability for the user equipment is 1.28 times and 1.15 times.

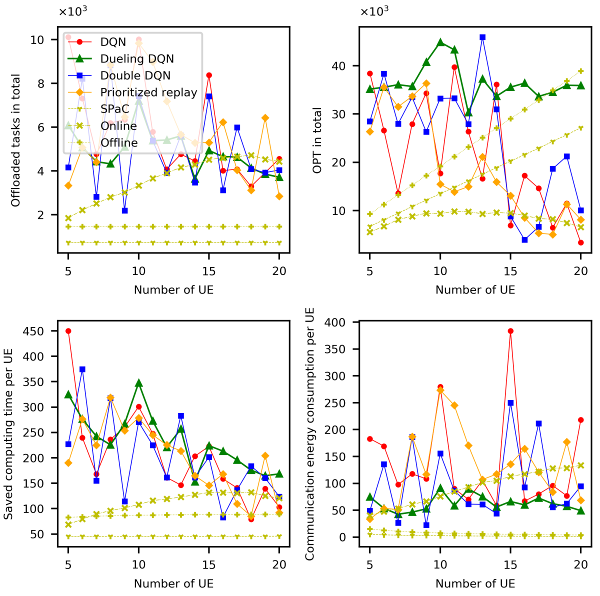


FIGURE 8. **UE cost and benefit as UE number increasing.**

Besides, as the UE number increases, the results obtained by Dueling DQN are always stable. The shortcoming of the exploration is the cause of unstable results. Since the RL method starts with random exploration, DQN algorithms would get good results, luckily, if they randomly choose a more favorable exploration direction. In the next section, we would analyze the reasons for the differences with more details.

C. SERVER LOAD AND PROFIT

As the UE number increases, the results obtained by Dueling DQN are always stable in FIGURE 9. Dueling DQN still occupies lower bandwidth during the decision-making

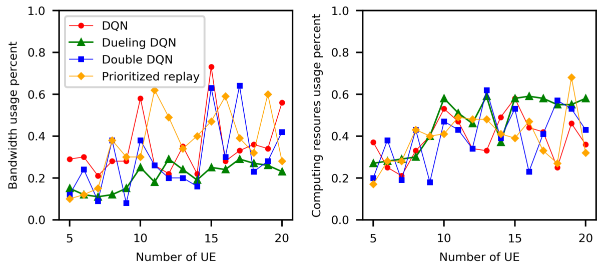


FIGURE 9. **Bandwidth usage(left) and computing resource usage(right) as UE number increasing.**

process to avoid bottlenecks in network transmission. For example, when the number of users is 18 and 19, both Dueling DQN and Double DQN spend the same computing resources, but Dueling DQN obtains better profit in FIGURE 8.

D. UE BEHAVIOR PREDICTION

In this section, we analyze the behavior prediction of UE with four RL methods. We focus on the choice of UE and their benefit. The consequences for offloading decisions are in the case of 5 and 20 UE is shown in Figure 10 and FIGURE 11. Among them, the horizontal axis is the UE ID, and the vertical axis is the optimization goal OPT.

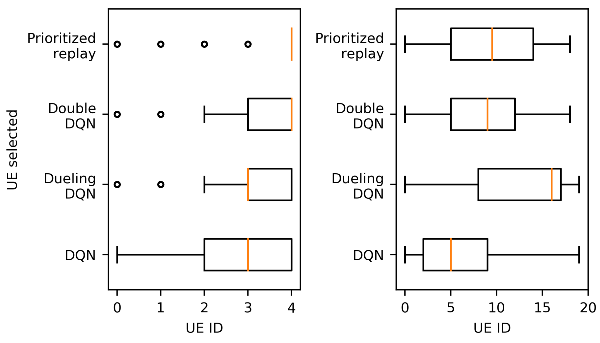


FIGURE 10. **UE selection for different RL algorithms of 5 UE(left) and 20 UE(right).**

According to the experimental design, the UE with a more significant ID number has a smaller amount of transmitted data size and a more amount of calculation performed. Intuitively, giving those UE more opportunities to offload would conducive to obtaining a better OPT. When there are 5 UE, all the algorithms show a UE selection tendency that is conducive to system optimization. Among them, the choice of Double DQN is more visible, while DQN tends to choose to improve the overall throughput. As the number of UE increasing to 20, Double DQN, Prioritized Replay, and DQN all make inevitable mistakes in UE selection. Therefore, Dueling DQN predicts UE behavior more accurately and improves offload throughput in both cases.

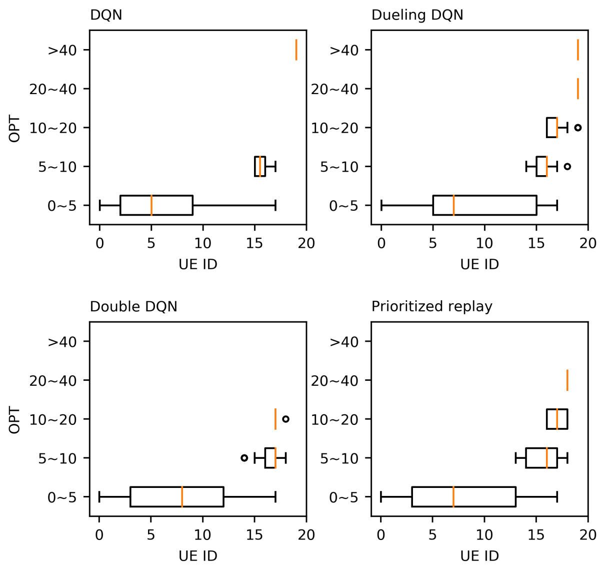


FIGURE 11. **UE benefit for different UE selection.**

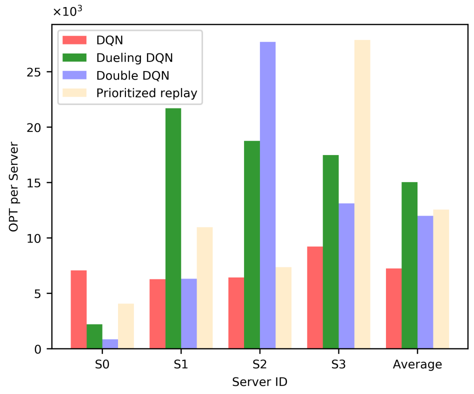


FIGURE 12. **Server benefit and balance in multiple server system.**

E. MULTIPLE SERVERS SCENARIO

In the final simulation, we set up 100 UE and four edge servers, see in TABLE 4. As server ID increasing, the computation rate is rising from 2 to 8 times. As UE ID k increasing, the amount of calculations for UE tasks is increasing as $\alpha_i^{(k)} = (k+1) \times 10$, and the amount of transmission data size is decreasing as $d_i^{(k)} = (n-k) \times 8$.

TABLE 4. **Multiple servers experiment parameter.**

Description	Value
Basic Transmitt Datasize	8KB
Total Bandwidth	50Mbps
Basic Running Time per Task	10ms
Basic Local Resource Obtain per Task	10%
Number of User Equipment	100
Number of servers	4
Computation Rate	2,4,6,8

In this simulation process, there are some common points from the extreme case of system resource exhaustion. For all RL algorithms, the average bandwidth occupancy rate of the four servers is close to 98.6%. The computing resource utilization rate is about 55.9%. The average number of offloading tasks completed is about 10547.

The difference is that under the same throughput and resource usage, the improved method of DQN achieves better optimization results OPT than DQN. Judging from the average of four servers (as shown in the final column), Dueling DQN is slightly better. Dueling DQN can better balance the load between servers. Under such an exhausting resource configuration, the bottleneck of edge services appears in the network transmission process. Therefore, balancing the utilization of resources is the key to optimizing complex systems.

Besides, for larger-scale edge computing systems, increasing server computing ability and network bandwidth are essential to ensure service quality. At the same time, increasing the number of neurons and network layers would improve the intelligence of the RL algorithm and require longer training time.

VII. CONCLUSION

We propose *soCoM* model in this paper to overcome the disadvantage of current works under the assumption that the end-users in the MEC system are diverse. *soCoM* uses a semi-online model and reinforcement learning to offload computation tasks according to the dynamic environments and loosely composed of the MEC systems. Experimental results show that Dueling DQN performs better for mass action space exploration in the computation offloading procedure. The methods also could be used to resolve large-scale computation offloading works in our future work.

REFERENCES

[1] X. Jiang, H. Shokri-Ghadikolaei, G. Fodor, E. Modiano, Z. Pang, M. Zorzi, and C. Fischione, "Low-latency networking: Where latency lurks and how to tame it," *Proc. IEEE*, vol. 107, no. 2, pp. 280–306, Feb. 2019.

[2] T. X. Tran, A. Hajisami, P. Pandey, and D. Pompili, "Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges," *IEEE Commun. Mag.*, vol. 55, no. 4, pp. 54–61, Apr. 2017.

[3] J. Tang, *Build 10+ Artificial Intelligence Apps Using TensorFlow Mobile and Lite for iOS, Android, and Raspberry Pi*. Birmingham, U.K.: Packt Publishing Ltd, 2018.

[4] M. Motamedi, D. Fong, and S. Ghiasi, "Machine intelligence on resource-constrained iot devices: The case of thread granularity optimization for CNN inference," *ACM Trans. Embedded Comput. Syst.*, vol. 16, no. 5, p. 151, 2017.

[5] H. Tang, J. Wang, L. Song, and J. Song, "Minimizing age of information with power constraints: Multi-user opportunistic scheduling in multi-state time-varying channels," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 5, pp. 854–868, May 2020.

[6] R. Xie, X. Jia, and K. Wu, "Adaptive online decision method for initial congestion window in 5G mobile edge computing using deep reinforcement learning," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 2, pp. 389–403, Feb. 2020.

[7] M. Chen and Y. Hao, "Task offloading for mobile edge computing in software defined ultra-dense network," *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 587–597, Mar. 2018.

[8] M. Avgeris, D. Dechouniotis, N. Athanasopoulos, and S. Papavassiliou, "Adaptive resource allocation for computation offloading: A control-theoretic approach," *ACM Trans. Internet Technol.*, vol. 19, no. 2, pp. 1–20, Apr. 2019.

[9] L. Tang and S. He, "Multi-user computation offloading in mobile edge computing: A behavioral perspective," *IEEE Netw.*, vol. 32, no. 1, pp. 48–53, Jan. 2018.

[10] C.-C. Lin, D.-J. Deng, Y.-L. Chih, and H.-T. Chiu, "Smart manufacturing scheduling with edge computing using multiclass deep Q network," *IEEE Trans. Ind. Informat.*, vol. 15, no. 7, pp. 4276–4284, Jul. 2019.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015.

[12] X.-Y. Zhang, F. Yin, Y.-M. Zhang, C.-L. Liu, and Y. Bengio, "Drawing and recognizing chinese characters with recurrent neural network," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 40, no. 4, pp. 849–862, Apr. 2018.

[13] Z. Wang, T. Schaul, M. Hessel, H. van Hasselt, M. Lanctot, and N. de Freitas, "Dueling network architectures for deep reinforcement learning," 2015, *arXiv:1511.06581*. [Online]. Available: http://arxiv.org/abs/1511.06581

[14] J. P. Champati and B. Liang, "Semi-online algorithms for computational task offloading with communication delay," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 1189–1201, Apr. 2017.

[15] S. Wang, H. Liu, P. H. Gomes, and B. Krishnamachari, "Deep reinforcement learning for dynamic multichannel access in wireless networks," *IEEE Trans. Cognit. Commun. Netw.*, vol. 4, no. 2, pp. 257–265, Jun. 2018.

[16] J. Li, H. Gao, T. Lv, and Y. Lu, "Deep reinforcement learning based computation offloading and resource allocation for MEC," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, Apr. 2018, pp. 1–6.

[17] 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.

[18] 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.

[19] M. Yan, G. Feng, J. Zhou, and S. Qin, "Smart multi-RAT access based on multiagent reinforcement learning," *IEEE Trans. Veh. Technol.*, vol. 67, no. 5, pp. 4539–4551, May 2018.

[20] 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.

[21] L. Huang, S. Bi, and Y. J. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Trans. Mobile Comput.*, early access, Jul. 24, 2019, doi: 10.1109/TMC.2019.2928811.

[22] Z. Ning, P. Dong, X. Wang, J. J. Rodrigues, and F. Xia, "Deep reinforcement learning for vehicular edge computing: An intelligent offloading system," *ACM Trans. Intell. Syst. Technol.*, vol. 10, no. 6, p. 60, 2019.

[23] Y. Liu, H. Yu, S. Xie, and Y. Zhang, "Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks," *IEEE Trans. Veh. Technol.*, vol. 68, no. 11, pp. 11158–11168, Nov. 2019.

[24] J. Wang, L. Zhao, J. Liu, and N. Kato, "Smart resource allocation for mobile edge computing: A deep reinforcement learning approach," *IEEE Trans. Emerg. Topics Comput.*, early access, Mar. 4, 2019, doi: 10.1109/TETC.2019.2902661.

[25] M. Hu, L. Zhuang, D. Wu, Y. Zhou, X. Chen, and L. Xiao, "Learning driven computation offloading for asymmetrically informed edge computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 8, pp. 1802–1815, Aug. 2019.

[26] F. Wang, J. Xu, X. Wang, and S. Cui, "Joint offloading and computing optimization in wireless powered mobile-edge computing systems," *IEEE Trans. Wireless Commun.*, vol. 17, no. 3, pp. 1784–1797, Mar. 2018.

[27] X. Chen, L. Jiao, W. Li, and X. Fu, "Efficient multi-user computation offloading for mobile-edge cloud computing," *IEEE/ACM Trans. Netw.*, vol. 24, no. 5, pp. 2795–2808, Oct. 2016.

[28] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, "A close examination of performance and power characteristics of 4G LTE networks," in *Proc. 10th Int. Conf. Mobile Syst., Appl., Services (MobiSys)*, 2012, pp. 225–238.

[29] S. Ohno, T. Shiraki, M. R. Tariq, and M. Nagahara, "Mean squared error analysis of quantizers with error feedback," *IEEE Trans. Signal Process.*, vol. 65, no. 22, pp. 5970–5981, Nov. 2017.

[30] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double Q-learning," in *Proc. 13th AAAI Conf. Artif. Intell.*, 2016, pp. 2094–2100.

[31] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," 2015, *arXiv:1511.05952*. [Online]. Available: http://arxiv.org/abs/1511.05952

[32] N. Matloff, "Introduction to discrete-event simulation and the simpy language," Dept. Comput. Sci., Univ. California Davis, Davis, CA, USA, Tech. Rep., Jan. 2008, vol. 2.

[33] 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 Comput. Archit. News*, vol. 45, no. 1, pp. 615–629, May 2017.

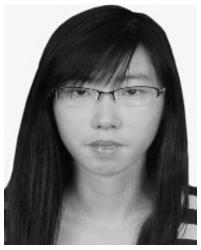
[34] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," in *Proc. Neural Inf. Process. Syst.*, 1999.

SHINAN SONG received the B.E. and M.S. degrees in computer science and technology from Jilin University, Changchun, China, in 2012 and 2015, respectively, where she is currently pursuing the Ph.D. degree. Her research interests include edge computing and deep neural networks.

ZHIYI FANG received the Ph.D. degree in computer science from Jilin University, Changchun, China, in 1998. He was a Senior Visiting Scholar with The University of Queensland, Brisbane, QLD, Australia, from 1995 to 1996, and with the University of California at Santa Barbara, Santa Barbara, CA, USA, from 2000 to 2001. He is currently a Professor of computer science with Jilin University, and a member of the China Software Industry Association and the Open System Committee of China Computer Federation. His research interests include distributed/parallel computing systems, mobile communication, and wireless networks.

ZHANYANG ZHANG (Member, IEEE) received the Ph.D. degree in computer science from The City University of New York (CUNY). He is currently a Tenured Professor with the Department of Computer Science, The City University of New York and a Doctoral Tutor. His main research areas involve wireless communication networks, underwater wireless sensor networks, network system simulation, and parallel/distributed computing.

CHIN-LING CHEN received the Ph.D. degree from National Chung Hsing University, Taiwan, in 2005. From 1979 to 2005, he was a Senior Engineer with Chunghwa Telecom Company, Limited. He is currently a Professor. His research interests include cryptography, network security, and electronic commerce. He has published more than 90 articles in SCI/SSCI international journals.



HONGYU SUN (Member, IEEE) received the Ph.D. degree from Jilin University, Changchun, China, in 2017. From January 2015 to September 2016, she was a Visiting Scholar with the University of Maryland, Baltimore County. She is currently an Assistant Professor with Jilin Normal University. She has published more than 20 articles in SCI/EI international conference proceedings and journals. Her research interests include wireless communication and mobile computing, RF-based sensing, privacy and security, and the Internet of Things (IoT).