IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2020.DOI

Collaborative Edge Computing and Caching with Deep Reinforcement Learning Decision Agents

JIANJI REN[1], HAICHAO WANG[1], TINGTING HOU[1], SHUAI ZHENG[1], CHAOSHENG TANG[1]
[1]College of Computer Science and Technology, Henan Polytechnic University, Jiaozuo, Henan, China.

Corresponding author: Jianji Ren (e-mail: renjianji@hpu.edu.cn).

ABSTRACT Large amounts of data will be generated due to the rapid development of the Internet of Things (IoT) technologies and 5th generation mobile networks (5G), the processing and analysis requirements of big data will challenge existing networks and processing platforms. As the most promising technology in 5G networks, edge computing will greatly ease the pressure on network and data processing analysis on the edge. In this paper, we considered the coordination between compute and cache resources between multi-level edge computing nodes (ENs), users under this system can offload computing tasks to ENs to improve quality of service (QoS). We aimed to maximize the long-term profit on the edge, while satisfying the low-latency computing of the users, and jointly optimize the edge-side node offloading strategy and resource allocation. However, it is challenging to obtain an optimal strategy in such a dynamic and complex system. To solve the complex resource allocation problem on the edge and make edge have certain adaptation and cooperation, we used double deep Q-learning (DDQN) to make decisions, ability to maximize long-term gains while making quick decisions. The simulation results prove the effectiveness of DDQN in maximizing revenue when allocation resources on the edge.

INDEX TERMS Collaborative computing, Edge computing, Optimization strategy

I. INTRODUCTION

As mobile communication and IoT technologies advances, smart cities, health care systems, etc. are deeply integrated with IoT technologies, and a large amount of data generated will pose challenges to data analysis and processing. Although the cloud computing [1] platform provides an efficient computing platform for big data processing, high bandwidth and high latency are unacceptable for scenarios with low latency requirements such as industrial control and real-time analysis.

In recent years, edge computing [2] as a new computing platform attracted the attention of researchers, although edge computing does not have a uniform definition, in essence, it is by deploying computing resources at the edge of the Internet, thereby reducing service delays, mitigating traffic pressure on the backhaul link and meeting the computational requirements of low latency applications.

Edge computing and cloud computing, distributed computing, parallel computing, etc. provide the necessary technical means to achieve accurate and fast integrated computing analysis. Cloud computing is based on platform virtualization, distributed storage, and parallel computing, flexible computing resources are allocated. Edge computing can be used as an extension of cloud computing [2], [3]. It provides ubiquitous and low latency and reliable computing.

However, edge computing has no powerful computing power of cloud computing. When a single computing node has many computing tasks, it is prone to high latency caused by long task queues. Therefore, edge computing still has great challenges in deployment and application.

(1) Firstly, the uncertainty of the computing task, due to the uncertainty of factors such as the size of the computing task, the length of computing time, and the delay of the task, the workload between edge computing nodes may vary greatly;

(2) Secondly, the workload scheduling of a single node, the task dynamic scheduling and computing resource allocation between nodes when collaborative computing between multiple nodes. The nodes of the same level have the same computing power, but there are differences in the number of tasks. It is of great significance to coordinate the workload balance between nodes and maintain low latency.

(3) Finally, the time interval, the most valuable part of edge

computing is the computing of low latency, so collaborative computing should meet the requirements of low latency.

To solve the above problems, in this work, we use deep reinforcement learning agents to determine the relevant nodes of collaborative computing. Specifically, we are using double deep Q-learning [4], [5] tomaximize the long-term profit of collaborative computing and ensure load balancing between nodes.

The rest of the paper is organized as follows: The second part summarizes the related work of collaborative computing in edge computing. The third part describes the dynamic system model of edge collaborative computing. The fourth part describes the collaborative computing strategy based on DDQN. We provided the results of simulation experiments and experiments in the fifth part. Finally, we summarized our work and discusses the direction of future work.

II. RELATED WORK

In recent years, edge computing networks based on multiple access have received extensive attention from academia and industry. Edge computing eliminates latency by providing a large number of computing resources for application services that require low latency and high computational demands. Although cloud computing has become very popular due to its powerful computing and flexible resource allocation strategies. However, due to the long distance between the end device and the cloud, cloud computing services may not provide assurance for low latency applications in the edge network.

To solve these problems, Edge Computing (EC) [2], [3], [6] has been studied to deploy computing resources closer to the user device, which can effectively improve applications' Quality of Service(QoS) that requires large amounts of computation and low latency. The computing of the task at the edge is complicated by the complex factors of computing, storage, caching, network, energy consumption, etc., it is difficult to make an offload strategy under low latency calculation limits, therefore, researchers used game theory to solve such problems. Zheng et al. [7] introduced random games to represent the mobile user's dynamic offload decision-making process and proposed a multi-agent random learning algorithm to solve the multi-user computing offload problem. Due to the problem of MEC multi-user computing offload in a multi-channel wireless interference environment, Chen et al. [8] proposed to use the game theory, and proves the advantages of the algorithm in energy consumption and computing execution time.

In addition, heuristic algorithms or dynamic programming methods can also be used to solve computational offloading problems. Dinh et al. [9] proposed a joint optimization computational offloading framework that can improve task allocation decisions and adjusts the CPU frequency of mobile devices. Mao et al. [10] proposes a dynamic calculation offload algorithm, which is based on Lyapunov optimization. This algorithm can jointly determine the CPU frequency and offload strategy of the energy harvesting equipment MECO problem.

For global model training, a sorted list network with multiple losses is proposed by Sheng et al. [11] to speed up training. This method can effectively mine training samples and avoid time-consuming initialization. An online orchestration framework that can be used for cross-edge service function chains is proposed by Zhou et al. [12], the framework can dynamically optimize the flow routing and resource allocation jointly to improve the overall cost efficiency as much as possible. In order to sample and improve network decisions in flow-aware software-defined networks, Wang et al. [13] proposed a space-time cooperative sampling (STCS) framework, and the experimental results prove the effectiveness of its sampling.

Recently, researchers have begun to use machine learning or deep learning to optimize the computational offload strategy for edge computing. Zhang et al. [14] proposed an intermittent connection cloudlet system based on the Markov decision process for the dynamic offloading problem of mobile users. But in the literature [15], in the mobile edge cloud, the author studied the dynamic service migration problem and proposed a sequential offload decision framework based on the Markov decision process.

Li et al. [16] proposed an RL-based optimization framework to solve the resource allocation problem in wireless MEC. The framework optimizes the offloading decision and computing resource allocation by optimizing the total cost of delay and energy consumption of all UEs. Yang et al. [17] proposed a computing resource allocation strategy based on deep reinforcement learning for URLLC edge computing networks with multiple users.

Wang et al. [18] considered the decision-making ability of reinforcement learning and the security of federated learning, a framework combining the two is proposed to optimize communication, caching, and computation on the edge side. Ren et al. [19] considered the dynamic workload and complex radio environment in the IoT environment, indicate the decision of the IoT device through multiple Deep Reinforcement Learning (DRL) agents, distributed training is performed on DRL agents through federated learning, and agents are also distributed on multiple edge nodes in a distributed manner.

For intelligent IoT applications, a framework is proposed by Liu et al. [20], which is based on the cloud edge architecture, apply federal learning to make smart applications available. In order to solve the heterogeneous problem in the IoT environment. Zhou et al. [21] made a comprehensive review of recent studies on EI. First, he reviewed and analyzed the motivation and background of artificial intelligence running at the edge of the network. Then he summarized several key technologies on the edge, deep learning frameworks, models, etc.. Edge intelligence builds intelligent edges by integrating DL into the edge computing framework to achieve dynamic and adaptive edge maintenance and management. Wang et al. [22] introduced and discussed the application scenarios of edge intelligence, methods and technologies used, and future work challenges.

The "Edge Artificial Intelligence" framework is designed [23] to intelligently use the collaboration between the device and the edge nodes to exchange data and model parameters, thereby better training and inferring the model. In order to deal with complex dynamic control problems, Wang et al. [24] proposed a FADE framework to accelerate training. Shen et al. [25] by deployed deep reinforcement learning (DRL) agents on IoT devices to make offload computing decisions and used federated learning (FL) to conduct distributed training for DRL. Wu et al. [26] proposed a hierarchical edge artificial intelligence learning framework HierTrain, which effectively deploys DNN training tasks on a hierarchical MECC structure.

When we consider communication, computing resource allocation, delay constraints, etc., the complexity of the edge computing system will be very high, it is challenging to obtain an optimal strategy in such a dynamic and complex system. Deep reinforcement learning is an improvement in reinforcement learning. The deep Q network is used to approximate the Q value function [4] to avoid excessively high estimates. It can be used to implement automatic resource allocation in wireless networks.

Therefore, we proposed that edge nodes use deep reinforcement learning agents to determine the allocation of computing resources and the maximum long-term benefits. Specifically, due to the complex resource allocation problem at the edge, we used DDQN as a decision agent, which makes the edge have certain adaptation and cooperation. Ability to maximize long-term gains while making quick decisions.

III. SYSTEM MODEL

This paper used the nodes with computing ability in the edge computing environment to analyze, as shown in Figure 1. Overall, the system is divided into four levels. The first is the device layer where the user device is located, including various networked devices of the user, such as mobile phones, IoT, VR, PC, etc., which establish connections with the Internet through a wireless network access point or 5G.

Secondly, the base station, cellular network, wireless network access point, etc., in which the user device is connected. These equipment are located at the edge of the Internet, connecting users and the Internet, and closest to the user device. Placing the edge computing nodes here will greatly reduce the delay and improve the users' experience, this paper assumes that the base station has an edge computing node which user connected to, and the node is marked as a level 1 computing node.

Then there is a level 2 compute node. The level 2 compute node is located between the level 1 compute node and the cloud computing platform of the core network. It acts as a collaborator for the compute node of the level 1 compute node. A level 2 compute node can coordinate a cache, calculation, etc. There are several levels 1 compute nodes in the area, and the level 2 compute nodes are close to the user, but not as close as the level 1 node.

Finally, the cloud computing platform is located in the core network. The cloud computing platform stores the running environment of the user application and the latest data and can release the file image to the computing node near the user when needed.

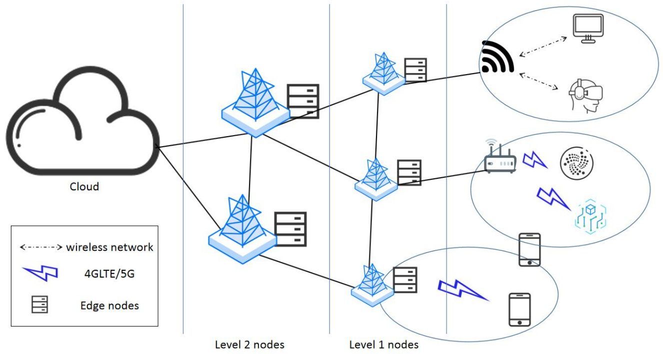

FIGURE 1: Edge computing supported IoT system

A. COMMUNICATION MODEL

The system includes $\boldsymbol{\beta}$ level 1 computing nodes and α level 2 computing nodes, wherein the level 1 computing node β belongs to $\boldsymbol{\beta}, \boldsymbol{\beta} = \{1, 2, ..., \beta\}$, and the level 2 computing node α belongs to $\boldsymbol{\alpha}, \boldsymbol{\alpha} = \{1, 2, ..., \alpha\}$. There are a total of $\boldsymbol{\gamma}, \boldsymbol{\gamma} = \{1, 2, ..., \gamma\}$ user devices, and user device γ belong to $\boldsymbol{\gamma}$. Among them, $\boldsymbol{\gamma}$ devices are divided into β groups, and user devices in each group are connected to β. For quantitative analysis, the time horizon is discretized into time epochs indexed by δ with equivalent duration as ϵ (in seconds).

We described the network model using a single base station β and the user device γ connected to it. When the device γ establishes a connection with the base station β, the base station allocates W Hz spectrum resources to the device, but the base station channel experiences a time-space change of rayleigh fading and following the flat fading model.

We denote ζ_δ^β as the channel gain during the epoch δ between the device and an EN $\beta \in \boldsymbol{\beta}$, which is assumed static and independently taken from a finite state space $\boldsymbol{\zeta}$. The ρ_δ^η is the transmit power with maximum limitation ρ_{max}^η, which Ψ is the power of interference plus noise. The transmission speed θ of the user device is calculated as follows:

$$\theta = W * log_2(1 + \zeta_\delta^\beta \cdot \rho_\delta^\eta / \Psi) \tag{1}$$

B. COMPUTING MODEL

We assume that every computing node in the system has the ability to be virtualization, and the application running environment image of the user device can be found in the cloud. Each level 1 compute node has an IP address and a remaining computing resource table of other level 1 nodes connected to it and a level 2 computing node α of the upper level. Within a certain period of time δ, the node connects to the surrounding node and one level 2 node α. The level

1 node β broadcasts its own remaining computing resource $\chi_\beta, \chi_\beta = (C_\beta, S_\beta)$ and accepts the remaining computing resource χ'_β broadcasts from other nodes, updating the local resource table based on the broadcast content.

We treat $\boldsymbol{\gamma}_\beta$ as a set of service requesters, where each requester γ belongs to $\boldsymbol{\gamma}_\beta$, connects to the nearest base station β according to its signal strength, and then sends a request to the compute node β where the base station is located, where $R_\beta^\gamma, R_\beta^\gamma = (D_s, T_l, C_r, S_r)$ is computing request send from user γ to the node β. Where D_s includes the task data and image file globally unique identifier (GUID), T_l is the computational delay limit of node β, C_r is the computing resource size required, and S_r is the storage resource size required.

After node β receives the task request R_β^γ, First check the remaining computing resources $\chi_\beta = (C_\beta, S_\beta)$ at the node β, and if the remaining computing resource χ_β meets the user computing requirement $C_r < C_\beta \& S_r < S_\beta$, then the service is started. During the service start phase, the node β searches for the image cache resource image required for the user task computing from the local cache area. If the image resource is in the local cache area, the image is loaded from the local cache area and the computing service is started. If there is no cached image locally, the node downloads the image file from the cloud platform to the node β and starts the computing service. When the calculation ends, the node β returns the computing result to the user device, completes the computing task of this period, waits for the user to compute the task for the next period or the user ends the computing command and pay for the task R_β^γ.

If the remaining computing resource χ_β at node β cannot satisfy the user computing requirement, $Cr > C_\beta \| S_r > S_\beta$, the service cannot be opened locally, and node β will forward the user request to the node in the computing resource table that meets the user's computing requirements. If the node x accepts the computing task, the user's computing service will be completed by the node x, and the base station β performs the transfer function here.

If there is no node in the calculation resource table that meets user requirements, the length of the queue determines whether the task is placed in the task queue or offloaded to the cloud. If the queue at node β is not full, $\phi_\beta < \phi_\beta^{max}$, the computing task is placed in the local task queue, and the task queue is a first in first out FIFO model; if the task queue at node β is full $\phi_\beta = \phi_\beta^{max}$, the computing task is offloaded to the cloud. In summary, the user computing task R_β^γ offload policy ω_β^γ is:

$$\omega_\beta^\gamma = \begin{cases} 0 & \text{if } C_r < C_\beta, S_r < S_\beta \text{ and } \phi_\beta = 0, \text{ node } \beta; \\ 1 & \text{if } C_r < C_x, S_r < S_x \text{ and } \phi_x = 0, \text{ node } x; \\ 2 & \text{if } C_r < C_x, S_r < S_\beta \text{ and } \phi_\beta < \phi_\beta^{max}, \text{ queue } \phi_\beta; \\ 3 & \text{Offload task } R_\beta^\gamma \text{ to the cloud.} \end{cases} \tag{2}$$

C. PAYMENT STRATEGY

After the user's computing request R_β^γ is completed by node β and the computing result is returned to the user device, the user device will pay to the node β according to the delay of the computing completion λ_δ^β. If the computing is completed within the limited time of T_l, the user pays according to the actual delay time. If the edge node times out to complete the computing task, users will not pay it.

$$F_\beta^\gamma = \begin{cases} \pi * \lambda_\delta^\beta & \text{if } \lambda_\delta^\beta < T_l, \pi \text{ is the price of edge;} \\ 0 & \text{if } \lambda_\delta^\beta > T_l; \\ \eta & \text{if task } R_\beta^\gamma \text{ failed, and } \eta < 0. \end{cases} \tag{3}$$

The delay λ_δ^β in this paper is defined as the time interval between when the user equipment initiates the calculation request and when the device receives the node calculation result. If the computing task is placed at the base station β to which the user device is connected, the computing delay λ_δ^β can be expressed as:

$$\lambda_\delta^\beta = \sigma_\gamma + \sigma_r^{'} + h_\beta + \sigma_\beta \tag{4}$$

Where σ_γ is the time spent on the transmit task R_β^γ, and $\sigma_r^{'}$ is the time it takes to transmit the result. h_β is the time taken by the computing node β to switch the computing task at the base station, which is a small fixed value.

$$\sigma_\gamma = D_s / \theta \tag{5}$$

σ_β is the time required for the computing node to complete the computing task. It is usually related to the CPU frequency f_{cpu}, the size of CPU cache c_{cpu}, and the data size D_s of the task data.

$$\sigma_\beta = D_s / (f_{cpu} * c_{cpu}) \tag{6}$$

If the computing task is placed at the neighboring base station β', the delay $\lambda_\delta^{\beta'}$ is:

$$\lambda_\delta^{\beta'} = \sigma_\gamma + \sigma_r^{'} + h_{\beta'} + \sigma_{\beta'} + 2 * d_\beta^{\beta'} / c \tag{7}$$

Where $d_\beta^{\beta'}$ is the fixed distance between the base station β and the neighbor node β', c is the speed of light. Finally, our objective function is:

$$\begin{aligned} &\max \sum_{\beta \in \{\boldsymbol{\beta}, \boldsymbol{\alpha}\}} \sum_{\delta} F_\beta^\gamma \\ &s.t. \ \forall \gamma \in \boldsymbol{\gamma}, \forall \beta \in \{\boldsymbol{\beta}, \boldsymbol{\alpha}\} \\ &\quad C_r >= 0, S_r >= 0 \end{aligned} \tag{8}$$

IV. COLLABORATIVE COMPUTING STRATEGY BASED ON DOUBLE DEEP Q-LEARNING

In order to better understand the DDQN agent, we briefly introduce DDQN in this paper. First, we introduce reinforcement learning. Reinforcement learning is an important branch of machine learning, agents in reinforcement learning can learn the actions that maximize the return through interaction with the environment. Unlike supervised learning, reinforcement learning does not learn from the samples provided. Instead, act and learn from their own experience in an uncertain environment.

Algorithm 1 Collaborative computing framework

```
Initialize:
Each edge node loads DDQN model as agent;
The compute node $\beta$ broadcasts its remaining computing power $\chi_\beta(C_\beta, S_\beta)$ to other nodes;
Node $\beta$ receives the remaining computing power broadcast and adds it to the calculation table $C$;
If: there is computing task $R_\beta^\gamma = (Ds, Tl, Cr, Sr)$ ;
The agent takes the node $\beta$ status $s$ and the task $R_\beta^\gamma$ as input, $s = C$ ;
Generate a decision policy according to action $a$ ;
Switch($a$): ;
    case 0: Immediately allocate computing resources and perform computing tasks;
    case 1: Put tasks into the local task queue $\phi$ and wait to allocate computing resources;
    case 2: Send task to the node in the computing resource table;
    case 3: Send tasks to the cloud computing platform;
Return: Send the results to the user device ;
The user pays the node according to the calculation delay and the task resource allocation amount ;
```

Reinforcement learning has two salient features: multiple trials and delayed rewards. Trial testing means weighing trade-offs between exploration and development. Agents will try some effective actions that can generate rewards based on past experience, but in order to generate higher returns, there is also a certain probability to explore new actions. Agents must take a variety of actions and gradually get the most out of it. Another feature of reinforcement learning is that agents should look at the global, not only considering immediate rewards, but also long-term cumulative rewards, which are designated as reward functions.

Model-free reinforcement learning has been successfully applied to the processing of deep neural networks and value functions [4]. It can directly use the original state representation as a neural network input to learn the strategy of difficult tasks [5]. Q-Learning is a model-free reinforcement learning algorithm. The most important component of the Q-learning algorithm is a method for correctly and effectively estimating the Q value. Q-functions can be implemented simply by look-up tables or function approximators, sometimes by nonlinear approximators, such as neural networks or even more complex deep neural networks. Q-learning is combined with deep neural networks, so-called Deep learning Q-learning(DQN). The formula for Q-learning is:

$$Q_\pi(s,a) = E[R_1 + \gamma R_2 + \cdots | S_0 = s, A_0 = a, \pi] \tag{9}$$

The parameter update formula is:

$$\theta_{t+1} = \theta_t + \alpha(Y_t^Q - Q(S_t, A_t; \theta_t)) \bigtriangledown_{\theta_t} Q(S_t, A_t; \theta_t) \tag{10}$$

Which Y_t^Q is defined as:

$$Y_t^Q = R_{t+1} + \gamma \cdot \underset{a}{max} Q(S_{t+1}, A_t; \theta_t) \tag{11}$$

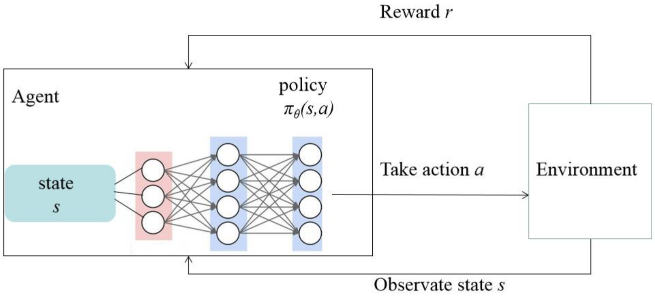

FIGURE 2: Decision Agent Based on DDQN

The formula of deep Q-learning is:

$$Y_t^{DQN} = R_{t+1} + \gamma \cdot \underset{a}{max} Q(S_{t+1}, a; \theta_t^{'}) \tag{12}$$

Improved DQN: double deep Q-learning. DDQN-based agent was shown in Figure 2. In conventional DQN, selecting an action and evaluating the selected action uses a maximum value that exceeds the Q value, which results in an overly optimistic estimate of the Q value. In order to alleviate the problem of overestimation, the target value in DDQN is designed and updated to

$$Y_t^Q = R_{t+1} + \gamma \cdot Q(S_{t+1}, \underset{a}{argmax} Q(S_{t+1}, a; \theta_t); \theta_t) \tag{13}$$

The error function in DDQN is rewritten as:

$$Y_t^{DoubleQ} = R_{t+1} + \gamma \cdot Q(S_{t+1}, \underset{a}{argmax} Q(S_{t+1}, a; \theta_t); \theta_t^{'}) \tag{14}$$

Among them, the action selection is separated from the target Q value generation. This simple technique makes the overestimation significantly reduced and the training process runs faster and more reliably.

Algorithm 2 Collaborative edge computing strategy based on double deep Q-learning

```
Initialization:
Initialize replay memory: $R$ and the memory capacity: $M$;
Main deep-Q network with random weights: $\theta$;
Target deep-Q network with weights: $\theta^- = \theta$;
For epoch $i$ in $I$:
    Input the system state $s$ into the generated Q-network;
    Compute the Q-value $Q(s, a; \theta)$ ;
    Input the system state $s'$ into the generated Q-network;
    Compute the Q-value $Q(s', a; \theta)$ ;
    Input the system state $s'$ into the target Q-network;
    Compute the Q-value $Q(s', a; \theta^-)$ ;
    Compute the target Q-value ;
        $Y = p(s, a) + \gamma Q(s', argmax(s', a; \theta), \theta^-)$ ;
    Output: action $a$ ;
    Record the changed status $s''$ and reward $F$ after action $a$ to memory $R$ ;
End For
Save model.
```

V. SIMULATION RESULTS

A. EXPERIMENT SETUP

This paper uses a simulation experiment method to instantiate user device and edge computing nodes for simulation through Python programming. The operating system used in the experiment was CentOS7, the processor was Intel E5-2650 V4, the hard disk size was 480G SSD + 4T enterprise hard disk, and the memory was 32G. The code interpreter is Python, version 3.6, and the code runtime dependencies include Tensorflow, Keras, Numpy, Scipy, Matplotlib, CUDA, etc..

The experimental data includes the computational task of the user offloading to the edge node in the time period i, which is randomly generated by calling the Bernoulli and Poisson functions in the Scipy library. The experiment assumes that the user device has the ability to connect to the network and can offload computing tasks and receive computing results.

Experiments in this paper compared Double deep Q-learning (DDQN), Deep Q-learning (DQN), Dueling deep Q-learning and Natural Q-learning, where the learning rate is set to 0.001, the replay memory size is 200, and the total training steps is 12000, the neural network update iteration cycle was set to update every 100 times. The penalty for mission failure is -30. After the task is successfully completed, agents will get rewards, the size of the reward obtained is closely related to the time the task is completed, the time the task is transmitted, and the total time spent offloading the calculation. In the experiment, the wireless channel was set to 6 different levels.

B. RESULT ANALYSIS

The reward obtained by the agent in the experiment is shown in Figure 3. During the period when training started, the neural network weights had just been initialized, and the agent could not give a good offload decision. At this time, the decision was randomly generated, resulting in The calculation of the received offload task fails and is punished, so the total reward is negative.

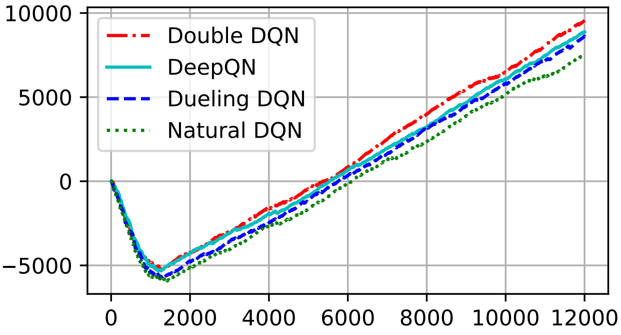

FIGURE 3: Rewards obtained by the agents during training.

However, with the increase of training and the update of neural networks, the agent can make the correct offload decision, and get more rewards than punishments, so the total rewards continue to increase with training. And it is obvious that agents based on DDQN can get more rewards with training.

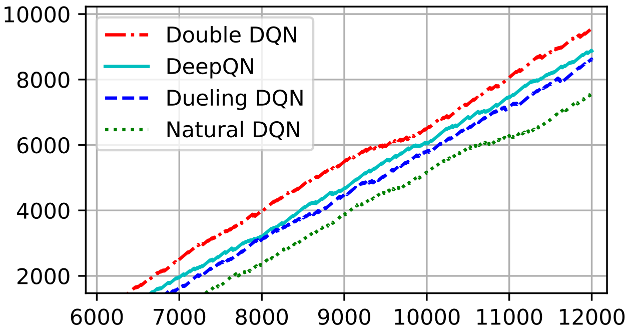

FIGURE 4: Detailed of rewards obtained by agents (6000-12000 steps).

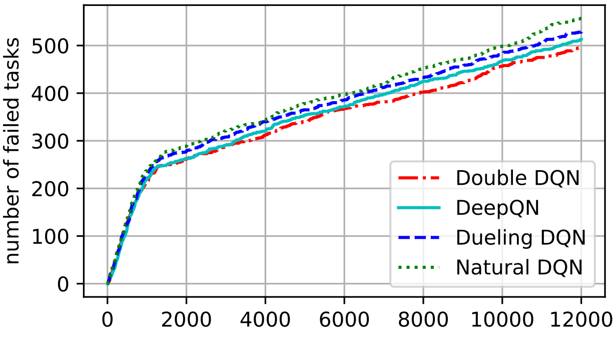

FIGURE 5: The number of failed tasks changes during agent training.

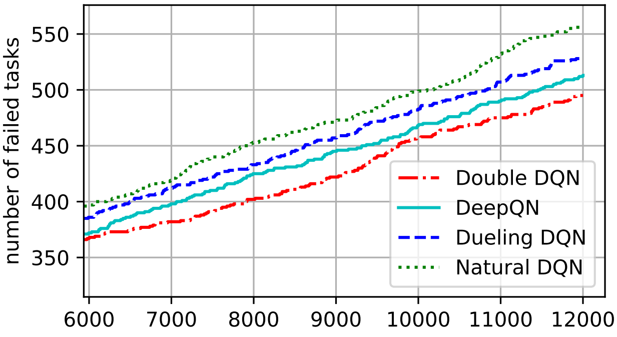

FIGURE 6: Detailed changes in the number of failed tasks(6000-12000 steps).

In order to more intuitively show the difference in rewards obtained by different neural network agents, the zoomed-in reward total changes are shown in Figure. 4.

Figure 5 shows the change in the number of task failures during the training process. At the beginning of the training,

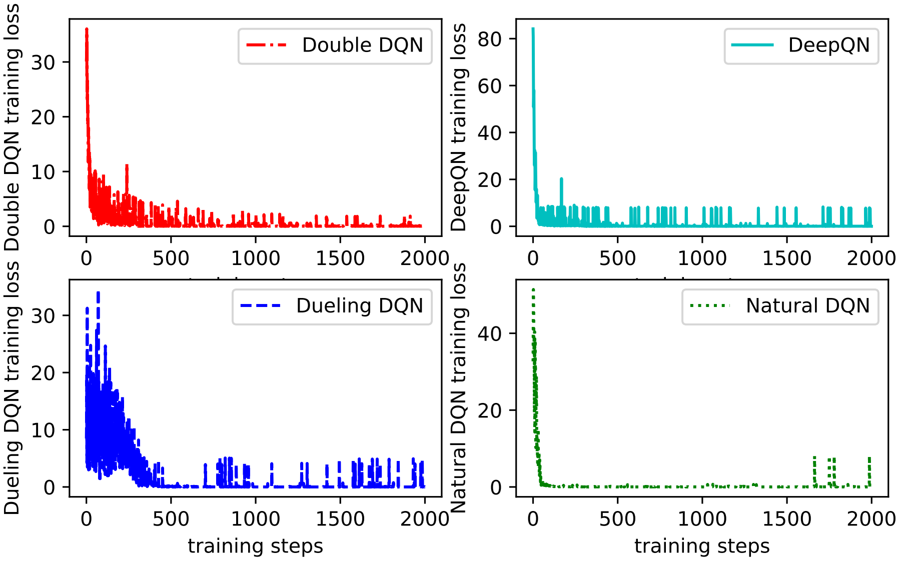

FIGURE 7: Agent training loss.

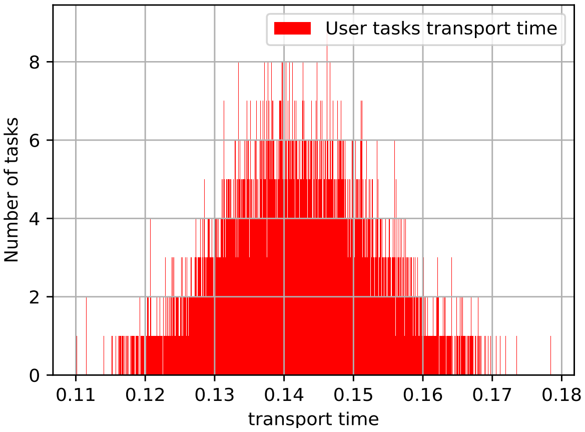

FIGURE 8: Distribution map of transmission time when task is offloaded.

the task calculation failed more, and the total number of task failures increased rapidly. With the training, the increased rate of task failures decreased and eventually reached a stable level. Observation shows that the DDQN-based agent has fewer task failures during the training process.

In order to more intuitively show the difference in the number of failed tasks of different neural network agents, the enlarged number change is shown in Figure. 6. The number of DDQN-based agent failures is always lower than other reinforcement learning.

Figure 7. shows the loss of change during training. The loss function of DDQN is defined as the square of the difference between the estimation function and the value function. The

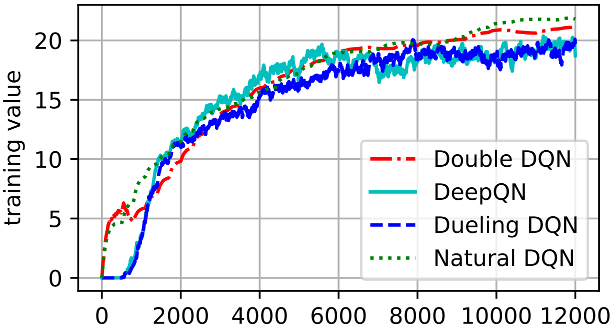

FIGURE 9: Value changes in agents.

loss is recorded every 6 pieces of training. The training loss based on DDQN is less than the other reinforcement learning at the beginning. As the weight of the neural network is updated, the loss is getting smaller and smaller, the training loss of several other reinforcement learning agents still fluctuates.

Record and display the user data transmission time in the experiment as shown in Figure 8. On the whole, the propagation time of user data is normally distributed, but it is irregular. In the experiment, some user data are more, but the network channel is poor. Therefore, the long transmission time leads to a high delay. However, some users have fewer data and the network is better, and the transmission time is shorter.

Figure 9 shows the value of the value calculated by the value function in the agent. At the beginning of the training, the value cannot be well estimated, but as the training progresses, the value estimation continues to approach the true level. And reached a stable level in the end.

VI. CONCLUSION AND FUTURE WORK

In this paper, we consider the bandwidth, computing, and cache resources of the ENs, benefit from the deep learning and powerful learning ability and decision-making characteristics, maximize the edge while satisfying the low-latency computing of users at the edge. In addition, it also considers the horizontal and vertical coordination cache and computing at the edge, which has certain adaptability and can fully coordinate the computing resources at the edge to maximize the value. However, the DDQN on the EN has a long training period and the effect is unstable. It needs to train for a while to make better decisions. In addition, when multiple ENs in the same group perform collaborative computing, we have not studied the prioritization strategy of computing resources or the computing resource bidding strategy. Future work we will focus on competitive bidding and allocation priorities. In addition, the security of users on the edge is also the focus of research.

VII. REFERENCE EXAMPLES

REFERENCES

[1] JoSEP A D, KAtz R A D, KonWinSKi A D, et al. A view of cloud computing[J]. Communications of the ACM, 2010, 53(4).

[2] Shi W, Cao J, Zhang Q, et al. Edge computing: Vision and challenges[J]. IEEE Internet of Things Journal, 2016, 3(5): 637-646.

[3] Satyanarayanan M. The emergence of edge computing[J]. Computer, 2017, 50(1): 30-39.

[4] Van Hasselt H, Guez A, Silver D. Deep reinforcement learning with double q-learning[C]//Thirtieth AAAI conference on artificial intelligence. 2016.

[5] Mnih V, Kavukcuoglu K, Silver D, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518(7540): 529.

[6] Hu Y C, Patel M, Sabella D, et al. Mobile edge computing:A key technology towards 5G[J]. ETSI white paper, 2015, 11(11): 1-16.

[7] Zheng J, Cai Y, Wu Y, et al. Stochastic computation offloading game for mobile cloud computing[C]//2016 IEEE/CIC International Conference on Communications in China (ICCC). IEEE, 2016: 1-6.

[8] Chen X, Jiao L, Li W, et al. Efficient multi-user computation offloading for mobile-edge cloud computing[J]. IEEE/ACM Transactions on Networking, 2015, 24(5): 2795-2808.

[9] Dinh T Q, Tang J, La Q D, et al. Offloading in mobile edge computing: Task allocation and computational frequency scaling[J]. IEEE Transactions on Communications, 2017, 65(8): 3571-3584.

[10] Mao Y, Zhang J, Letaief K B. Dynamic computation offloading for mobile-edge computing with energy harvesting devices[J]. IEEE Journal on Selected Areas in Communications, 2016, 34(12): 3590-3605.

[11] H. Sheng et al., "Mining Hard Samples Globally and Efficiently for Person Re-identification," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2020.2980549.

[12] Z. Zhou, Q. Wu and X. Chen, "Online Orchestration of Cross-Edge Service Function Chaining for Cost-Efficient Edge Computing," in IEEE Journal on Selected Areas in Communications, vol. 37, no. 8, pp. 1866-1880, Aug. 2019, doi: 10.1109/JSAC.2019.2927070.

[13] X. Wang, X. Li, S. Pack, Z. Han and V. C. M. Leung, "STCS: Spatial-Temporal Collaborative Sampling in Flow-Aware Software Defined Networks," in IEEE Journal on Selected Areas in Communications, vol. 38, no. 6, pp. 999-1013, June 2020, doi: 10.1109/JSAC.2020.2986688.

[14] Zhang Y, Niyato D, Wang P. Offloading in mobile cloudlet systems with intermittent connectivity[J]. IEEE Transactions on Mobile Computing, 2015, 14(12): 2516-2529.

[15] Wang S, Urgaonkar R, Zafer M, et al. Dynamic service migration in mobile edge-clouds[C]//2015 IFIP Networking Conference (IFIP Networking). IEEE, 2015: 1-9.

[16] Li J, Gao H, Lv T, et al. Deep reinforcement learning based computation offloading and resource allocation for MEC[C]//2018 IEEE Wireless Communications and Networking Conference (WCNC). IEEE, 2018: 1-6.

[17] Yang T, Hu Y, Gursoy M C, et al. Deep reinforcement learning based resource allocation in low latency edge computing networks[C]//2018 15th International Symposium on Wireless Communication Systems (ISWCS). IEEE, 2018: 1-5.

[18] Wang X, Han Y, Wang C, et al. In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning[J]. IEEE Network, 2019, 33(5): 156-165.

[19] Ren J, Wang H, Hou T, et al. Federated Learning-Based Computation Offloading Optimization in Edge Computing-Supported Internet of Things[J]. IEEE Access, 2019, 7: 69194-69201.

[20] D. Liu, X. Chen, Z. Zhou and Q. Ling, "HierTrain: Fast Hierarchical Edge AI Learning with Hybrid Parallelism in Mobile-Edge-Cloud Computing," in IEEE Open Journal of the Communications Society, doi: 10.1109/OJCOMS.2020.2994737.

[21] Z. Zhou, X. Chen, E. Li, L. Zeng, K. Luo and J. Zhang, "Edge Intelligence: Paving the Last Mile of Artificial Intelligence With Edge Computing," in Proceedings of the IEEE, vol. 107, no. 8, pp. 1738-1762, Aug. 2019, doi: 10.1109/JPROC.2019.2918951.

[22] X. Wang, Y. Han, V. C. M. Leung, D. Niyato, X. Yan and X. Chen, "Convergence of Edge Computing and Deep Learning: A Comprehensive Survey," in IEEE Communications Surveys & Tutorials, vol. 22, no. 2, pp. 869-904, Secondquarter 2020, doi: 10.1109/COMST.2020.2970550.

[23] X. Wang, Y. Han, C. Wang, Q. Zhao, X. Chen and M. Chen, "In-Edge AI: Intelligentizing Mobile Edge Computing, Caching and Communication by Federated Learning," in IEEE Network, vol. 33, no. 5, pp. 156-165, Sept.-Oct. 2019, doi: 10.1109/MNET.2019.1800286.

[24] X. Wang, C. Wang, X. Li, V. C. M. Leung and T. Taleb, "Federated Deep Reinforcement Learning for Internet of Things with Decentralized Cooperative Edge Caching," in IEEE Internet of Things Journal, doi: 10.1109/JIOT.2020.2986803.

[25] Shihao Shen, Yiwen Han, Xiaofei Wang, and Yan Wang. 2019. Computation Offloading with Multiple Agents in Edge-Computing–Supported IoT. ACM Trans. Sen. Netw. 16, 1, Article 8 (February 2020), 27 pages. DOI:https://doi.org/10.1145/3372025

[26] Q. Wu, K. He and X. Chen, "Personalized Federated Learning for Intelligent IoT Applications: A Cloud-Edge based Framework," in IEEE Computer Graphics and Applications, doi: 10.1109/OJCS.2020.2993259.

JIANJI REN is currently an associate professor in College of Computer Science and Technology, Henan Polytechnic University. He received M.S. and Ph.D. degrees from the School of Computer Science and Engineering, Dong-A University in 2007 and 2010 respectively. He received the B.S. degree in the Department of Mathematics, JiNan University in 2005. His current research interests are mobile content-centric networks, collaborative caching in edge computing.

HAICHAO WANG received the B.S. degree in natural geography and resource environment from Henan Polytechnic University, Jiaozuo, Henan, China, in 2018. He is currently pursing the software engineering Master's degree, college of computer science and technology(software college), Henan Polytechnic University, Jiaozuo, Henan, China. His research interests include edge computing, edge caching, big data analysis, deep learning and the Internet of things technology.

TINGTING HOU is current a B.S. in college of computer science and technology from Henan Polytechnic University, Jiaozuo, Henan, China. Her current major interests include edge computing, edge caching, deep learning, big data analysis and the Internet of things technology.

SHUAI ZHENG is currently a B.S. in college of computer science and technology, Henan Polytechnic University, Jiaozuo, Henan, China. His current major interests include edge computing, edge caching, deep learning, big data analysis and data mining.

CHAOSHENG TANG received PH.D. in Management Science and Engineering from Yanshan University, Qinhuangdao, Hebei, China, in 2015. His major interests are machine learning, complexity theory, multimedia applications, and online social networks.