# Adaptive Collaborative Computing in Edge Computing Environment

Jianji Ren, Haichao Wang, and Xiaohong Zhang(✉)

Henan Polytechnic University, Jiaozuo 454000, Henan, China
xhzhpuedu@163.com

**Abstract.** The rapid development of 5th generation mobile networks (5G) and Internet of Things (IoT) technologies will generate a large amount of data, 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 consider 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 aim 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. Therefore, we use double deep Q-learning (DDQN) to make decisions to solve the complex resource allocation problem on the edge and make edge have certain adaptation and cooperation. 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.

**Keywords:** Collaborative computing · Edge computing · Optimization strategy

## 1 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 paltform 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,


X. Wang et al. (Eds.): 6GN 2020, LNICST 337, pp. 160–172, 2020.
https://doi.org/10.1007/978-3-030-63941-9_12

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; (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] to minimize the delay 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. We provide the results of simulation experiments and experiments in the fourth part. Finally, the fifth part summarizes our work and discusses the direction of future work.

## 2 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, because the distance between the cloud and the end device is typically large, 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 require 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 use game theory to solve such problems. Zheng et al. [7] expressed the dynamic offloading decision process of mobile users as a random game and proposed an efficient multi-agent random learning algorithm to solve the multi-user computing offloading problem. Chen et al. [8] proposed a game-based computational offloading algorithm as a solution for MEC multi-user computing offloading in multi-channel wireless interference environments, with excellent performance in terms of energy consumption and computational 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] proposed a dynamic computational offloading algorithm based on Lyapunov optimization, which can jointly determine the offloading strategy and CPU frequency for the MECO problem of energy harvesting equipment.

Recently, researchers have begun to use machine learning or deep learning to optimize the computational offload strategy for edge computing. Zhang et al. [11] proposed an intermittent connection cloudlet system based on Markov decision process for the dynamic offloading problem of mobile users. But in the literature [12], the author studies the dynamic service migration problem in the mobile edge cloud and proposes a sequential offloading decision framework based on the Markov decision process.

Li et al. [13] 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. [14] proposed a computing resource allocation strategy based on deep reinforcement learning for URLLC edge computing networks with multiple users.

Wang et al. [15] combined deep reinforcement learning and federated learning frameworks with mobile edge systems to optimize mobile analysis edge computing, caching and communication. Ren et al. [16] considering 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 deployed on multiple edge nodes, and use the federated learning (FL) to train DRL agents in a distributed manner.

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 advanced reinforcement learning algorithm that uses a deep Q network to approximate the Q-value action function [4] and has been utilized in wireless networks to achieve automatic resource allocation.

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, we use DDQN to make decisions to solve the complex resource allocation problem on the edge and make edge have certain adaptation and cooperation. Ability to maximize long-term gains while making quick decisions.

## 3 System Model

This paper uses the nodes with computing ability in the edge computing environment to analyze, as shown in Fig. 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., which the user device is connected. These equipments 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.

### 3.1 Communication Model

The system includes $M$ level 1 computing nodes and $N$ level 2 computing nodes, wherein the level 1 computing node $m$ belongs to $M, M = \{1, 2, ..., M\}$, and the level 2 computing node $n$ belongs to $N, N = \{1, 2, ..., N\}$. There are a total of $D, D = \{1, 2, ..., D\}$ user devices, and user device $d$ belong to $D$. Among them, $D$ devices are divided into $M$ groups, and user devices in each group are connected to $M$. For quantitative analysis, time horizon is discretized into time epochs indexed by $i$ with equivalent duration as $\phi$ (in seconds).

We described the network model using a single base station $m$ and the user device $d$ connected to it. When the device $d$ establishes a connection with the base station $m$, the base station allocates $W$ Hz spectrum resources to the device,

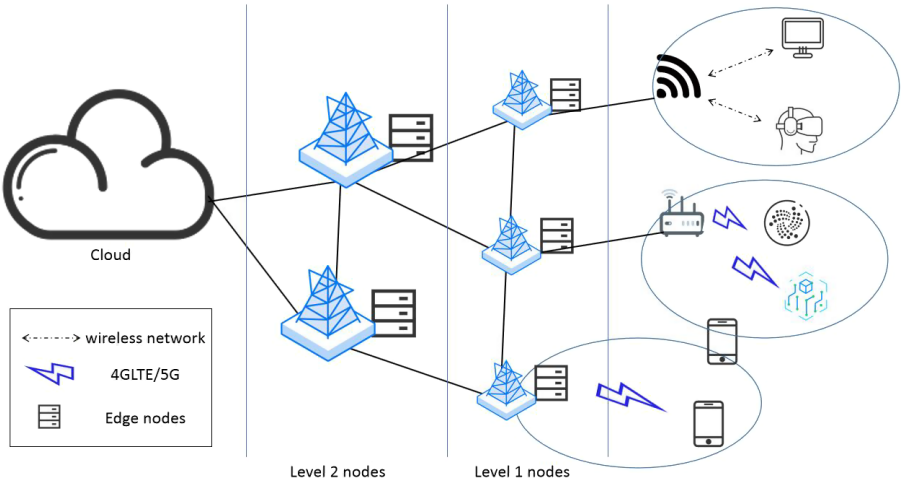


**Fig. 1.** System architecture diagram.

but the base station channel experiences a time-space change of rayleigh fading, and following the flat fading model.

We denote $s_i^m$ as the channel gain during the epoch $i$ between the device and an EN $m \in M$, which is assumed static and independently taken from a finite state space $S_e$. The $f_i^{tr}$ is the transmit power with maximum limitation $f_{max}^{tr}$, which $A$is the power of interference plus noise. The transmission speed $v$ of the user device is calculated as follows:

$$v = W * log_2(1 + s_i^m \cdot f_i^{tr}/A) \tag{1}$$

### 3.2 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 $n$ of the upper level. Within a certain period of time $i$, the node connects to the surrounding node and one level 2 node $n$. The level 1 node $m$ broadcasts its own remaining computing resource $p_m, p_m = (C_m, S_m)$ and accepts the remaining computing resource $p$ broadcasts from other nodes, updating the local resource table based on the broadcast content.

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

After node $m$ receives the task request $R_m^d$, First check the remaining computing resources $p_m = (C_m, S_m)$ at the node $m$, and if the remaining computing resource $p_m$ meets the user computing requirement $C_r < C_m \& S_r < S_m$, then the service is started. During the service start phase, the node $m$ 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 cache image locally, the node download the image file image from the cloud platform to the node $m$ and start the computing service. When the calculation ends, the node $m$ 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 $R_m^d$.

If the remaining computing resource $p_m$ at node $m$ cannot satisfy the user computing requirement, $Cr > C_m \| S_r > S_m$, the service cannot be opened locally, and node $m$ 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 $m$ performs the transfer function here.

If there is no node in the calculation resource table that meets user requirement, 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 $m$ is not full, $l_m < l^{max}$, 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 $m$ is full $l_m = l^{max}$, the computing task is offloaded to the cloud. In summary, the user computing task $R_m^d$ offload policy $p_m^d$ is:

$$p_m^d = \begin{cases} 0 & \text{if } C_r < C_m, S_r < S_m \text{ and } l_m = 0, \text{ Computed at node } m; \\ 1 & \text{if } C_r < C_x, S_r < S_x \text{ and } l_x = 0, \text{ Computed at node } x; \\ 2 & \text{if } C_r < C_x, S_r < S_m \text{ and } l_m < l^{max}, \text{ Waiting in node } m; \\ 3 & \text{Offload task } R_m^d \text{ to the cloud.} \end{cases} \tag{2}$$

### 3.3 Payment Strategy

After the user's computing request $R_m^d$ is completed by node $m$ and the computing result is returned to the user device, the user device will pay to the node $m$ according to the delay of the computing completion $del_t^m$. If the computing is completed within the limited time $T_l$, 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.

$$fe_m^d = \begin{cases} \pi * del_t^m & \text{if } del_t^m < T_l, \pi \text{ is the price of edge;} \\ 0 & \text{if } del_t^m > T_l; \\ \eta & \text{if task } R_m^d \text{ failed, and } \eta < 0. \end{cases} \tag{3}$$

The delay $del_t^m$ 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 $m$ to which the user device is connected, the computing delay $del_t^m$ can be expressed as:

$$del_t^m = tr_d + tr_{res} + h_m + ct_m \tag{4}$$

Where $tr_d$ is the time spent on the transmit task $R_m^d$, and $tr_{res}$ is the time it takes to transmit the result. $h_m$ is the time taken by the computing node m to switch the computing task at the base station, which is a small fixed value.

$$tr_d = D_s/v \tag{5}$$

$ct_m$ is the time required for the computing node to complete the computing task. It is usually related to the CPU frequency $fr_{CPU}$, the size of CPU cache $B_{ca}$, and the data size $D_s$ of the task data.

$$ct_m = D_s/(fr_{CPU} * B_{ca}) \tag{6}$$

If the computing task is placed at the neighboring base station $x$, the delay $del_t^x$ is:

$$del_t^x = tr_d + tr_{res} + h_x + ct_x + 2 * dis_m^x/c \tag{7}$$

Where $dis_m^x$ is the fixed distance between the base station $m$ and the neighbor node $x$, $c$ is the speed of light. Finally, our objective function is:

$$\begin{aligned} &\max \sum_{m \in \{M,N\}} \sum_i fe_m^d \\ &s.t. \forall d \in D, \forall m \in \{M,N\} \\ &\quad C_r >= 0, S_r >= 0 \end{aligned} \tag{8}$$

## 4 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 $m$ broadcasts its remaining computing power $P(C_m, S_m)$ to other nodes;
Node $m$ receives the remaining computing power broadcast and adds it to the calculation table $C$;
**If:** there is computing task $R_m^d = (Ds, Tl, Cr, Sr)$ ;
The agent takes the node $m$ status $s$ and the task $R_m^d$ 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 $q$ 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 tend to use the effective actions they have tried in the past to generate rewards, but it must also explore better new actions to generate higher returns in the future. 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:

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

The formula of deep Q-learning is:

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

Improved DQN: double deep Q-learning. 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.**

## 5 Simulation Results

### 5.1 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.

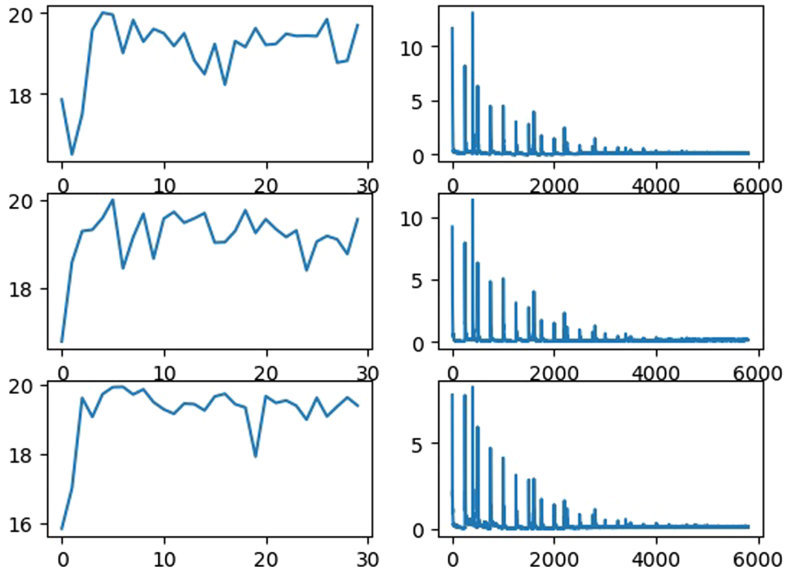


**Fig. 2.** The result graph shows the training utility (left) and loss (right) changes of random three nodes.

### 5.2 Result Analysis

In Fig. 2, the learning curve of agents are shown above, the probability of a user offloading a task is 0.5, the utility and loss of three nodes are randomly selected as the display. The left side is the utility of the node, the abscissa is the training period, a total of 30 cycles are trained, each cycle includes 200 training steps, and each step of the training will get a utility. After averaging the utility of each cycle, the utility curve of the agents is got. The utility graph shows that with the training of the agent, the edge node's profit gradually increases in a certain period of time, but as the training increases, the growth rate of utility begins to

decrease. On the right is the training loss map. The loss is defined as the mean square error between the target value and the predicted value. A total of 6000 steps are trained, each step will have some fluctuation due to the update of the weight of the neural network. However, from the trend point of view, the loss is gradually reduced.

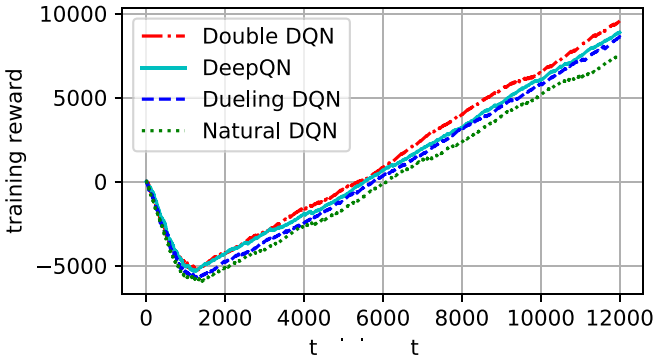


**Fig. 3.** Rewards obtained by the agents during training.

The reward obtained by the agent in the experiment is shown in Fig. 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.

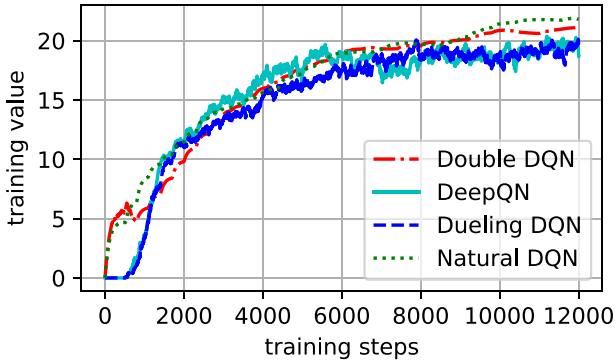


**Fig. 4.** Value changes in agents.

Figure 4 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.

## 6 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.

## References

1. Josep, A.D., Katz, R.A.D., Konwinski, A.D., et al.: A view of cloud computing. Commun. ACM **53**(4), 50–58 (2010)
2. Shi, W., Cao, J., Zhang, Q., et al.: Edge computing: vision and challenges. IEEE Internet Things J. **3**(5), 637–646 (2016)
3. Satyanarayanan, M.: The emergence of edge computing. Computer **50**(1), 30–39 (2017)
4. Van Hasselt, H., Guez, A., Silver, D.: Deep reinforcement learning with double Q-learning. In: Thirtieth AAAI Conference on Artificial Intelligence (2016)
5. Mnih, V., Kavukcuoglu, K., Silver, D., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529 (2015)
6. Hu, Y.C., Patel, M., Sabella, D., et al.: Mobile edge computing? A key technology towards 5G. ETSI White Pap. **11**(11), 1–16 (2015)
7. Zheng, J., Cai, Y., Wu, Y., et al.: Stochastic computation offloading game for mobile cloud computing. In: 2016 IEEE/CIC International Conference on Communications in China (ICCC), pp. 1–6. IEEE (2016)
8. Chen, X., Jiao, L., Li, W., et al.: Efficient multi-user computation offloading for mobile-edge cloud computing. IEEE/ACM Trans. Netw. **24**(5), 2795–2808 (2015)
9. Dinh, T.Q., Tang, J., La, Q.D., et al.: Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans. Commun. **65**(8), 3571–3584 (2017)
10. Mao, Y., Zhang, J., Letaief, K.B.: Dynamic computation offloading for mobile-edge computing with energy harvesting devices. IEEE J. Sel. Areas Commun. **34**(12), 3590–3605 (2016)
11. Zhang, Y., Niyato, D., Wang, P.: Offloading in mobile cloudlet systems with intermittent connectivity. IEEE Trans. Mob. Comput. **14**(12), 2516–2529 (2015)
12. Wang, S., Urgaonkar, R., Zafer, M., et al.: Dynamic service migration in mobile edge-clouds. In: 2015 IFIP Networking Conference (IFIP Networking), pp. 1–9. IEEE (2015)
13. Li, J., Gao, H., Lv, T., et al.: Deep reinforcement learning based computation offloading and resource allocation for MEC. In: 2018 IEEE Wireless Communications and Networking Conference (WCNC), pp. 1–6. IEEE (2018)

14. Yang, T., Hu, Y., Gursoy, M.C., et al.: Deep reinforcement learning based resource allocation in low latency edge computing networks. In: 2018 15th International Symposium on Wireless Communication Systems (ISWCS), pp. 1–5. IEEE (2018)
15. Wang, X., Han, Y., Wang, C., et al.: In-edge AI: intelligentizing mobile edge computing, caching and communication by federated learning. IEEE Netw. **33**(5), 156–165 (2019)
16. Ren, J., Wang, H., Hou, T., et al.: Federated learning-based computation offloading optimization in edge computing-supported internet of things. IEEE Access **7**, 69194–69201 (2019)