International Journal of Digital Crime and Forensics
Volume 15 • Issue 1

Task Offloading in Cloud-Edge Environments:
A Deep-Reinforcement-Learning-Based Solution

Suzhen Wang, Hebei University of Economics and Business, China
Yongchen Deng, Hebei University of Economics and Business, China*
Zhongbo Hu, Hebei University of Economics and Business, China

ABSTRACT

Cloud computing involves transferring data to remote data centers for processing, which consumes significant network bandwidth and transmission time. Edge computing can effectively address this issue by processing tasks at edge nodes, thereby reducing the amount of data transmitted and enhancing the utilization of network bandwidth. This paper investigates intelligent task offloading under the three-layer architecture of cloud-edge-device to fully exploit the cloud-edge collaboration potential. Specifically, an optimization objective function is constructed by modelling the processing cost of all computing tasks. Additionally, asynchronous advantage actor-critic (A3C) algorithm is proposed under cloud-edge collaboration to solve the optimization problem of minimizing the sum of the weights of task offloading delay and energy consumption. Experimental results indicate that the algorithm can effectively utilize the computing resources of the cloud center, reduce task execution delay and energy consumption, and compare favourably with three existing task offloading methods.

KEYWORDS

A3C, Cloud-Edge Collaboration, Deep Reinforcement Learning, Task Offload

INTRODUCTION

The fast-paced development of big data and related technologies has led to a tremendous increase in the scale of data (Warren, 2015). The conventional cloud computing mode (Zhao et al., 2018) needs to transfer all computational tasks to the cloud server for processing, which causes response delay, energy loss, data security and other problems in the transmission process. Edge computing technology has experienced rapid growth as a means to address the limitations of cloud computing technology. Edge computing (Shi et al., 2016; Tran et al., 2017), and specifically mobile edge computing or MEC (Khan et al., 2019), is a groundbreaking data processing methodology that involves processing data closer to the edge of network-connected devices. When servers are in close proximity to mobile device

DOI: 10.4018/IJDCF.332066 *Corresponding Author

users, the wireless LAN can be utilized to deliver necessary services and computing capabilities. MEC offers assistance for applications that are sensitive to delay, require low latency, are mobile, and need to be location-aware.

However, despite these positives, edge nodes have restricted computing resources and processing capacity. The amount of computing resources that edge nodes assign to mobile users is determined by their "workload," or the total count of simultaneous tasks unloaded to the edge servers. When a significant number of portable devices unload task requirements to the same edge server, the workload will be higher and the processing time delay of the job will grow. Therefore, the optimization of the unloading strategy (Mao, You, et al., 2017; Shakarami et al., 2020; Sun et al., 2010; Zheng et al., 2019) can reduce the processing delay of the unloading task and enhance the level of service and user-friendliness of the device.

In the MEC network, there are not many studies on cloud-edge collaboration. The majority of research on MEC computing offloading has focused on distributed task allocation and resource management among mobile gadgets (Alam et al., 2018) and between mobile devices and MEC servers (Li et al., 2019). However, there has been a lack of emphasis on fully leveraging the computing resources and processing power of cloud servers. If the full computing power of the cloud center and the distributed characteristics of edge computing can be extracted to optimize the fine-grained computing task unloading strategy, this will not only relieve the dimension explosion caused by unloading a large number of subtasks to multiple edge servers but also decrease the total cost of the task unloading system. The focus of this paper is to introduce a method for fine-grained task unloading that relies on deep reinforcement learning. The three main ideas are:

1. From the basis of the three-tier cloud-edge architecture, a fine-grained task unloading model is proposed in the cloud-edge framework. The optimization aim of this model is minimizing the sum of "task offloading delay and energy consumption" weights. In the model, the edge server assumes responsibility for the local optimization of task offloading policies, and the cloud center is in charge of global optimization. The cloud server and edge nodes work together to achieve distributed optimization of task offloading.
2. With regard to the terminal task for partial offloading at the edge, the terminal task is split into several subtasks, and the interdependency between the subtasks is expressed through a directed acyclic graph to form a task division model.
3. A novel approach is suggested to facilitate cloud-edge collaboration by combining the asynchronous dominant actor-critic algorithm with a task division model using the A3C algorithm. By leveraging the benefits of both deep learning and reinforcement learning, this algorithm offers a promising solution to mitigate end-task offloading delays and system energy consumption.

AN ANALYSIS OF RELEVANT LITERATURE

MEC research categorizes task offloading into two distinct types: complete offloading (Mao, Zhang, et al., 2017; Zhan et al., 2020) and partial offloading (Chae et al., 2017; Eshraghi & Liang, 2019; Tran & Pompili, 2017). Complete offloading means that the user's computational tasks are atomic in nature and cannot be further partitioned, and the entire user task is processed by the terminal or offloaded to the edge server for processing. Alam et al. (2018) proposed an independent management framework utilizing Q-learning technology. To minimize latency and save energy utilization, Li et al. (2019), considering the service cache, introduced the opportunistic network into the multiaccess network. A suboptimal algorithm utilizing the sequence game was developed to address the issue by considering delay as well as energy usage as the total computing overhead. Partial offloading means that the endpoint task can be partitioned; the task is processed by first dividing the task into different subtasks and then making the corresponding offloading decisions for these subtasks. In a partially offloaded multiuser MEC system, Mao, Zhang, et al. (2017) created an algorithm that manages joint

radio and computational resources. To address the issue of partial offloading policies when users withhold personal information, Zhan et al. (2020) created a distributed computational offloading algorithm. Chae et al. (2017) proposed a multiuser offloading algorithm that uses convex optimization to lower the amount of energy used by a mobile device with latency constraints on the energy usage of mobile gadgets. Despite their effectiveness, these solutions overlook the task dependency among end users and only concentrate on determining the optimal workload allocation for the MEC server.

Due to the distributed deployment feature of edge servers, the distributed approach can solve the task offloading problem more effectively than the traditional centralized task offloading algorithm (Chen & Hao, 2018; Eshraghi & Liang, 2019; Poularakis et al., 2019; Tran & Pompili, 2017). The asynchronous advantage actor-critic algorithm (A3C) is a special deep reinforcement learning algorithm that combines value functions and policy methods, where actors generate actions by policy methods, which are then evaluated by commentators based on value functions; multiple actors and commentators interact in various contexts to guide the intelligent agent to generate optimal action policies. Tang & Wong (2022) examine the dynamic load of edge nodes, which is not known in advance. Each mobile device can make a decision about offloading in a decentralized manner based on locally perceived information. The literature (Chen et al., 2016) joins task offloading and computational service collaboration problems, considering collaboration between edge servers and remote clouds as well as collaboration between edge servers. Zhao et al. (2019) construct an integrated model of computing offloading and resource allocation as a binary optimization problem, translate the problem into an equivalent form of reinforcement learning, and propose a distributed deep learning algorithm to achieve weighting and minimize cost. All of these studies use distributed offloading methods, without considering partial offloading task types.

Considering the shortcomings of previous research in this field, this paper investigates the offloading mechanism of collaborative tasks at the cloud edge based on deep reinforcement learning. First, by comprehensively considering computational resources, time delay, and energy consumption, an optimization process is carried out to minimize both task execution delay and energy consumption. Second, a subtask partitioning model is established based on the dependency relationship between tasks, and the partitioned model is combined with the A3C algorithm to propose an asynchronous dominant actor-critic algorithm to solve this optimization problem. This algorithm leverages the computational capabilities of both the cloud and edge computing environments to make optimal offloading decisions accurately and quickly, in line with the diversity and dynamic change characteristics of the environment in which the edge nodes are located.

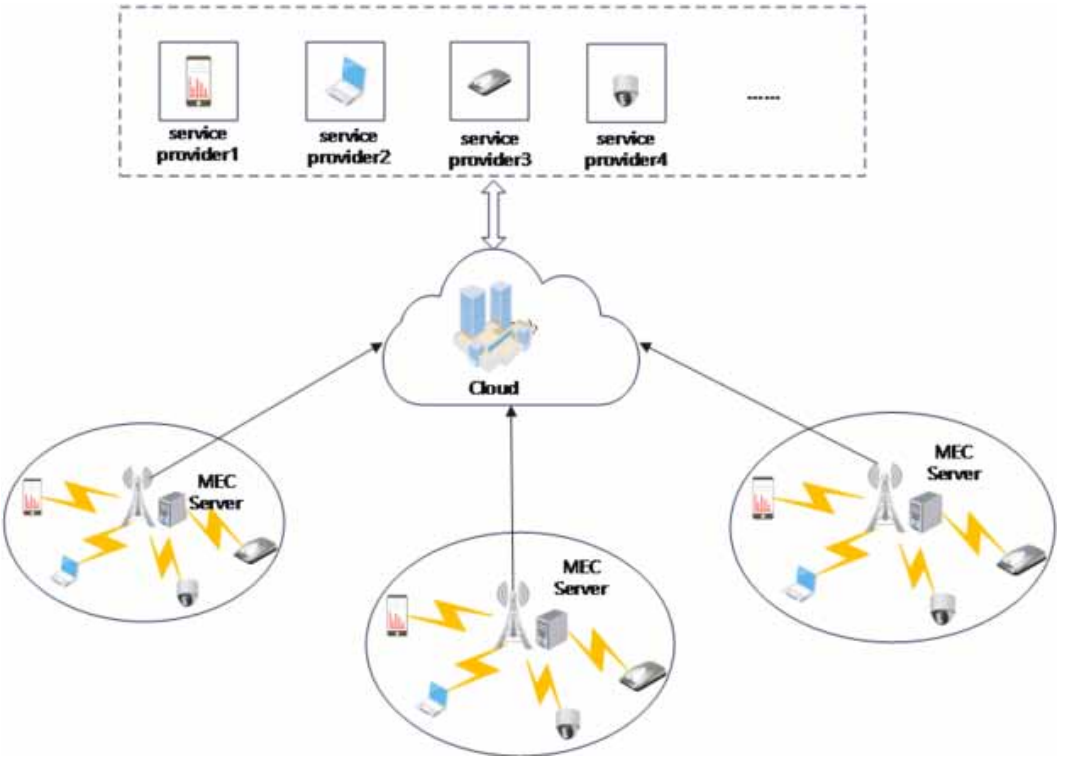
CLOUD-EDGE-END SYSTEM MODEL CONSTRUCTION

This section covers the formal definition of the cloud-edge-end system architecture, the construction of a computational model for the cloud-edge-end system, and the optimization of target parameters (latency and energy consumption) for the cloud-edge-end system.

Formal Definition of Architecture

This paper puts forward a cloud-edge-device collaboration task offloading model. The entire system model structure includes the user layer, edge layer, and cloud layer, as shown in Figure 1. The user layer consists of M user terminals. The user set is $M = \{1 \ldots m \ldots M\}$. The edge layer contains P edge nodes, and each edge node consists of a wireless signal receiver (AP) and an edge server. The AP is responsible for collecting information about mission requirements and network conditions of end-users within its coverage area. The set of edge servers is $P = \{1 \ldots p \ldots P\}$. In the cloud layer, the cloud center is located in a layer where the edge nodes and cloud servers are generally connected through wired channels. Some special calculation tasks may consume more resources and take a

Figure 1. System architecture

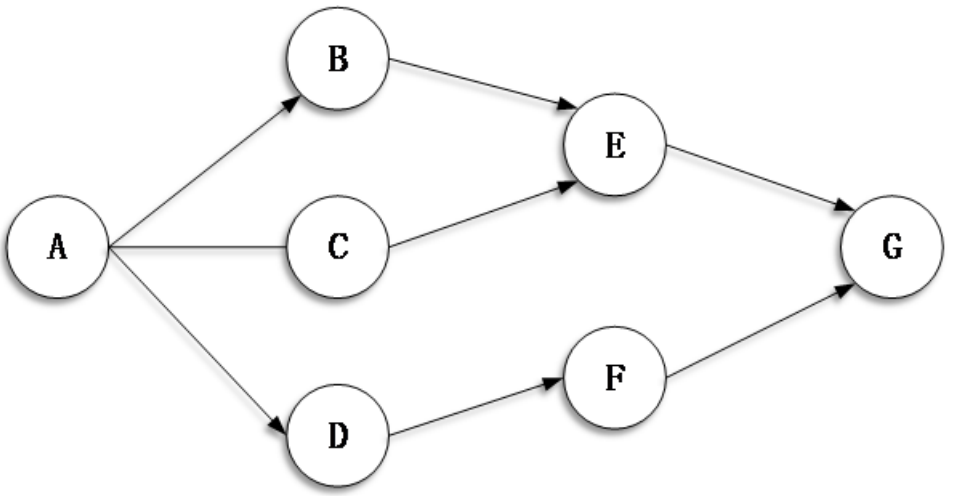


longer time, such as data backup in large companies. Such tasks can be transferred to a cloud server for processing and are therefore not in the scope of the model.

The terminal application computing task is divided into N subtasks, denoted by $N = \{1 \ldots n \ldots N\}$. The subtask requirements of the end-user can be symbolized by a directed acyclic graph (DAG), which has a start node and an end node representing the initial task and termination task, as shown in Figure 2, respectively. The initial task and the termination task are usually local tasks because the former needs to be obtained from the local device and the latter needs to return the output to the local device. The user task is represented as a weighted DAG, designed as $G(V, E)$ where V represents the subtask $V = \{v_i, i \in N\}$, E represents the directed edge between nodes, and the directed edge $e(i, j)$ represents the priority relationship between subtasks, so that task v_i should be completed before v_j starts.

In this system, time is discretized into different time periods. In the early stages of each time slot, the AP collects the task requirements within its coverage and sends them to the edge server, which makes the unloading decision for the subtasks divided by users according to the task requirements, evaluates the unloading decision made, and constantly adjusts the unloading strategy in line with the merits of the evaluation. The process of the edge server adjusting its unloading strategy is called learning strategy optimization knowledge. The different edge nodes at the edge layer send the learned strategy optimization knowledge to the cloud center, which summarizes the received optimization knowledge, obtains the optimized unloading strategy, and sends it back to each edge server for implementation. The optimization process of the end user task unloading strategy is carried out

Figure 2. Example: Subtask dependency diagram



asynchronously and simultaneously on multiple edge servers, and it is finally summarized by the cloud center to meet the distributed characteristics.

Computational Model Construction

In this system, it is important to consider task processing delay as well as energy consumption when deciding whether to execute the user's subtasks locally or offload them to the edge server. Let $x^l_{m,n}$ and $x^e_{m,n,p}$ represent that subtask *n* of user *m* is processed locally and offloaded to the edge server for processing, respectively. $x^l_{m,n} \in \{0,1\}$ and $x^e_{m,n,p} \in \{0,1\}$. $x^l_{m,n} = 1$ implies that user *m* selects to process subtask *n* locally, and $x^e_{m,n,p} = 1$ means that user m selects to offload subtask *n* to the p th edge server for processing. We use $X = \{x_{m,n} \mid m \in M, n \in N\}$ to denote the set of offloading strategies for all subtasks of all end users.

Local Processing

When subtask n of user m is executed locally, without any task unloading, the cost of the system includes only local processing delay and local energy consumption. Let $d_{m,n}$ denote the quantity of computational input data related to subtask n of terminal m, and $c_{m,n}$ denote the computational workload required to complete subtask n. Then, $t^l_{m,n}$ is the completion time of subtask n locally executed on device m:

$$t^l_{m,n} = \frac{c_{m,n}}{f^l_{m,n}} \tag{1}$$

In Equation 1, $f^l_{m,n}$ is the computing resource allocated by mobile device m to subtask n.

Let $E^l_{m,n}$ be the power usage of handheld device m to process subtask n locally. The energy consumption model is:

$$E_{m,n}^{l} = k\left(f_{m}^{l\,2}\right)\ c_{m,n} \tag{2}$$

In Equation 2, k depends on the energy coefficient of the chip structure, and we set $k = 10^{-11}$.

For user m subtask n to be completed, all preceding tasks (or pretasks) must first be processed, since there is interdependence between subtasks. Let $TR_{m,n}^{l}$ denote the time taken by subtask *n* of user *m* to be processed locally. Additionally, $TF_{m,k}^{l}$ represents the time taken to complete the *k*th predecessor task of subtask n on the local server, while $TF_{m,k}^{e}$ denotes its completion time on edge servers. $k \in pre\left(n\right)$ denotes all immediate predecessors of subtask n. Therefore, $TR_{m,n}^{l}$ can be represented as:

$$TR_{m,n}^{l} = \max_{k \in pre(n)} \max\{TF_{m,k}^{l}, TF_{m,k}^{\mathrm{e}}\} \tag{3}$$

This paper does not take into account the delay of returning the calculation result to the end user through the edge server. Thus:

$$TR_{m,n}^{l} \geq \left(1 - x_{m,k}^{e}\right) TF_{m,k}^{l} + x_{m,k}^{e} TF_{m,k}^{p}, k \in pre\left(n\right) \tag{4}$$

This means that subtask *n* can only be executed after task *k* has been completed. However, this does not factor in the time it takes for the wireless channel to receive the return result of the subtask.

The time taken to locally process user *m*'s subtask *n* is the sum of the time taken to prepare the subtask and the time taken to actually process the task locally. This can be represented by:

$$TF_{m,n}^{l} = t_{m,n}^{l} + TR_{m,n}^{l} \tag{5}$$

Thus, the processing cost of subtask *n* of portable device *m* calculated locally is:

$$C_{m,n}^{l} = \alpha TF_{m,n}^{l} + \left(1 - \alpha\right) E_{m,n}^{l} \tag{6}$$

In Equation 6, α represents the factor that assigns weight to time, while $(1 - \alpha)$ represents the factor that assigns weight to energy consumption for subtasks. The weight factors can differ across various application scenarios and exhibit varying degrees of dependence on both time delay and energy consumption.

MEC Processing

In the cloud edge network discussed in this paper, when a portable device *m* offloads the subtask *n* to the edge node for processing, the total delay consists of two parts: the uplink transmission delay of the terminal task, and the processing delay of the MEC node. When the subtask uploads the calculation data to the edge server, orthogonal frequency division multiplexing (OFDM) is used to ensure that the interference between the coverage areas of two adjacent edge nodes is properly managed. For each edge node, only one mobile device in its coverage area can use a specific OFDM subchannel at any given time. Let $r_{m,n,p}$ indicate the uplink transmission speed between user *m* and edge server *p*:

$$r_{m,n,p} = Wlog_2\left(1+\frac{P^p_{m,n}H^p_{m,n}}{I+\sigma^2_{m,p}}\right) \tag{7}$$

In Equation 7, W is the channel bandwidth, $P^p_{m,n}$ is the transmission power of user *m* offloading its subtask n to edge node p, and the default setting is 27dBm. $H^p_{m,n}$ is the channel gain of user *m* transmitting subtask n to edge node p, $H^p_{m,n} = \left(d^p_m\right)^{-\eta}\left|h^p_m\right|^2$, where d^p_m is the Euclidean distance between user m and edge server p, η is a constant set to 4 (Chen et al., 2018), and h^p_m is the corresponding Rayleigh fading channel. The interference from other edge nodes is represented by I, and the noise power is $\sigma^2_{m,p}$.

Building on this, if $t^{trans}_{m,n}$ represents the uplink transmission delay of the terminal task, then:

$$t^{trans}_{m,n} = \frac{d_{m,n}}{r_{m,n,p}} \tag{8}$$

Furthermore, if $t^p_{m,n}$ represents the process time of the edge server and $f^p_{m,n}$ represents the computing resources allocated by MEC server p to subtask n, then:

$$t^{exe}_{m,n} = \frac{c_{m,n}}{f^p_{m,n}} \tag{9}$$

The process time model of the edge node is calculated as:

$$t^e_{m,n} = t^{trans}_{m,n} + t^{exe}_{m,n} \tag{10}$$

The energy consumption associated with processing tasks on edge servers can be divided into two components: the energy expenditure during data transmission, and the energy expenditure during server computing. The data transmission energy expenditure is:

$$E^{trans}_{m,n} = P^p_{m,n}\ t^{trans}_{m,n,p} \tag{11}$$

The energy consumption is:

$$E^{exe}_{m,n} = c_{m,n}\theta_p \tag{12}$$

where θ_p represents the energy expenditure coefficient of edge server p. Therefore, the total energy expenditure model of the edge server is:

$$E^e_{m,n} = E^{trans}_{m,n} + E^{exe}_{m,n} \tag{13}$$

Let $TR^{e}_{m,n}$ denote the time when subtask n of user m is prepared to be processed on the edge server:

$$TR^{e}_{m,n} = \max\{TF^{trans}_{m,n}, \max_{k\in pre(n)} TF^{e}_{m,k}\} \tag{14}$$

Here, $TF^{trans}_{m,n}$ represents the time taken for the transmission from user m to the edge server to be completed:

$$TF^{trans}_{m,n} = t^{trans}_{m,n} + \max_{k\in pre(n)} TF^{l}_{m,k} \tag{15}$$

In simpler terms, the time taken to complete the task transfer is the sum of the transfer time of the current subtask plus the time taken to locally complete all previous subtasks of the current subtask. If the previous task *k* of subtask *n* is executed locally, $TF^{e}_{m,k} = 0$. Therefore, $\max_{k\in pre(n)} TF^{e}_{m,k}$ is the time taken to unload all direct predecessors of subtask *n* processed in the edge server. Consequently, the duration required to complete subtask *n* for user *m* on the edge server is equal to the combined time it takes to prepare for the subtask and process it on the server:

$$TF^{e}_{m,n} = t^{exe}_{m,n} + TR^{e}_{m,n} \tag{16}$$

Then, the processing cost of offloading subtask n of wireless device m to the edge node is:

$$C^{e}_{m,n} = \alpha TF^{e}_{m,n} + \left(1-\alpha\right) E^{e}_{m,n} \tag{17}$$

Target Parameter Optimization

The subtasks divided by the end user have dependence. V_N represents the termination task of end user m, and the completion time of subtask V_N is the completion delay of whole computing tasks of user m. Let T_m denote the total task completion time of end user m. Therefore,

$$T_m = TF_{m,N} \tag{18}$$

The optimization goal is to find the most efficient computing offloading strategy in order to reduce the overall cost of task completion for all users over a period of time. With this in mind, let C_m denote the total cost of task processing for end user m:

$$C_m = \sum_{n=1}^{N} C_{m,n} = \sum_{n=1}^{N} (x^{l}_{m,n} C^{l}_{m,n} + x^{e}_{m,n} C^{e}_{m,n}), \forall n \in N \tag{19}$$

Then, the problem of minimizing the cost of processing all computing tasks under the cloud-edge-terminal system architecture can be modeled as:

$$\min_{X} \sum_{m=1}^{M} C_m, \forall m \in M \tag{20}$$

s.t. C1: $x_{m,n}^{l} + x_{m,n,p}^{e} \leq 1$,

C2: $x_{m,n}^{l} \in \{0,1\}, x_{m,n,p}^{e} \in \{0,1\}$,

C3: $T_m \leq T_{m,max}$,

C4: Equation (4); C5: Equation (14)

Constraint C1 guarantees that subtask n can only be executed on local or edge servers, constraint C2 controls the binary, and constraint C3 controls the completion time, indicating that the completion delay of user m 's termination subtask is no greater than the user's maximum allowable time. $T_{m,max}$ is determined by the delay requirements of the end user. Execution can only begin after all its predecessor tasks have been completed. Constraint C4 controls the dependence of the local task processing, indicating that subtask n can only start executing after all its immediate predecessors have been completed. Constraint C5 controls the dependence of the edge server in processing tasks, meaning that the execution can only start when the subtask is completely transferred to the edge node.

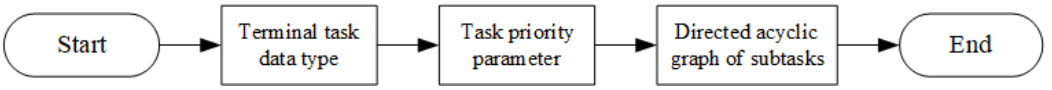
SUBTASK MODEL AND ALGORITHM OPTIMIZATION

This paper focuses on the optimization of task processing costs using an NP-hard nonconvex integer programming problem. Since this cannot be effectively solved using traditional methods, it requires the use of heuristic algorithms. Deep reinforcement learning is an online intelligent learning method that adjusts policies and maximizes rewards through the interaction between agents and the environment. The A3C algorithm is a special deep reinforcement learning method that uses the idea of asynchronous multithreading to: break the correlation of data, interact with its environment in multiple threads, and regularly upload its learning experience to the public network for parameter updates. Therefore, this paper proposes the use of an asynchronous dominant actor-critic algorithm under cloud-edge collaboration (CEC-A3C). Compared to the centralized task offloading method, which is not very effective, the CEC-A3C algorithm maximizes the use of the distributed characteristics of the cloud-edge network architecture. This will more effectively guide edge nodes to interact with the environment and solve the problem of dimensionality explosion caused by the fine-grained division of end-users.

Subtasking Model

The overall architecture of the CEC-A3C scheme mainly includes two parts: the environment layer, and the algorithm layer. This section will explore the first part—the environment layer—in more detail, while the algorithm layer is explored in the following section. The environment layer corresponds to the user layer in the system architecture. Its main function is that the end-user divides the computing task into more fine-grained subtasks with certain dependence constraints, and it then sends the offload request to the edge layer. Choosing an efficient and feasible task division method in the environment layer is a key issue since existing task division models are often too complicated. Therefore, this paper proposes graph theory and the hierarchical division method as a better solution. For the primary selection classification, tasks with roughly the same type are the same set. The tasks in the task set are assigned with task parallelism and task priority parameters of the subtasks. Under this rule, the greedy strategy is used to generate the task division result, with a directed acyclic graph $G(V, E)$ between subtasks. The task division process is shown in Figure 3:

Figure 3. Task division process



CEC-A3C Algorithm

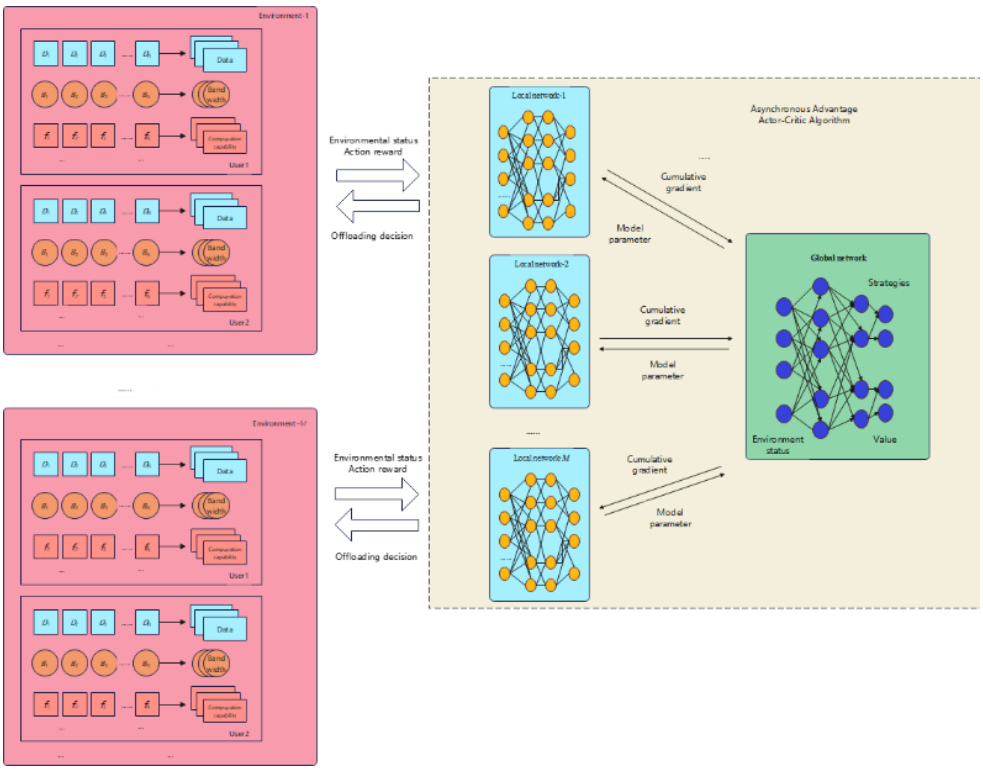
The algorithm layer corresponds to the edge layer and cloud layer in system architecture, including in the CEC-A3C algorithm. The specific execution process, which is shown in Figure 4, follows these steps:

1. The wireless receiver AP in the edge node receives the environmental information, including input data volume and transmission power of each subtask.
2. The edge server, as an agent, interacts with the environment to which it belongs, and it makes the unloading decision of the mobile terminal's subtask.
3. The nodes send the learned gradient parameters to the cloud, and they regularly receive new parameters from the cloud. This helps them to more effectively guide the current edge node and the subsequent environment to learn and interact

CEC-A3C Algorithm Elements

State Space. Most research on edge computing task offloading using deep reinforcement learning defines the state space as the task requirements of all end-users in the environment, network conditions,

Figure 4. CEC-A3C algorithm process



and computing resources of edge servers within a certain period (Dong et al., 2019; Liu et al., 2019). However, using the methods explored in this paper, the fine-grained division of terminal tasks will cause the number of subtasks after division to be large and dependent. The traditional state-space division method will cause problems such as a high state space dimension, and difficult convergence of the algorithm. Therefore, this paper proposes using a task selection method based on task sets, so that the agent only observes the task offloading demand state of one end user at a time, thereby reducing the dimension of the state space.

Initially, the AP collects the task requirements of all users in the environment and creates a task set. The edge server randomly selects a task requirement from the task set, extracts the status of the task requirement, and selects the unloading strategy of each subtask by the agent. After the task processing is completed, the subchannel resources are released, and the task requirement is deleted from the task set. The recorded completion time of the current task unloading decision is the time when the next task request starts to be processed. This operation is looped until all end-user tasks in the task set are processed. If the system state of the current user subtask at time *t* is defined as a multidimensional vector, then the state of the environment to which agent p belongs at time t is:

$$S_p(t) = \{ \bigcup_{m \in M'}^{M'} m, D_m^{total}, [d_{m,1}, \cdots d_{m,N}], b_m^{total}, [b_{m,1}, \cdots b_{m,N}], [f_{m,1}, \cdots f_{m,N}] \} \tag{21}$$

In Equation 21,

- M' is the user task set in the environment to which agent *p* belongs,
- m is the task requirement currently processed by the agent,
- D_m^{total} is the total data size of the task,
- $\left[d_{m,1}, \cdots d_{m,N}\right]$ is the subtask data size,
- $\left[f_{m,1}, \cdots f_{m,N}\right]$ is the quantity of computing resources allocated by the edge server to the subtask,
- b_m^{total} is the total number of channels allocated by the current processing user, and
- $\left[b_{m,1}, \cdots b_{m,N}\right]$ is the number of channels allocated by subtasks.

Action space. In this paper, the action space of the system state is defined as the manner in which the agent allocates channel bandwidth and quantity of computing resources to the user subtasks. This can be expressed as:

$$A(t) = \left(x_t^{m,n}, \lambda_t^{m,n}, \eta_t^{m,n}\right) \tag{22}$$

In Equation 22,

- $x_t^{m,n}$ represents the agent's selection of subsequent subtasks at time t,
- $\lambda_t^{m,n}$ represents the subchannel bandwidth of subtask n at time t, and
- $\eta_t^{m,n}$ represents t the computational resources allocated by the edge server to subtask n at time t.

Reward Function. After performing possible actions in state space S_t, the agent will receive rewards, the function of which is related to that of the objective. The goal of task unloading is to

optimize the sum cost of all terminal tasks. Therefore, the reward $\mathrm{r}_{\mathrm{t}}^{\mathrm{a}}$ obtained by the edge node when performing action a in system state S_{t} is the calculation cost generated through the current task unloading decision compared to the cost optimized when the task is processed locally. This can be expressed as:

$$r_t^a = \alpha\left(t_{m,n}^l - t_{m,n}\right) + \left(1-a\right)\left(e_{m,n}^l - e_{m,n}\right) \tag{23}$$

If $\mathrm{R}\left(\mathrm{t}\right)$ represents the cumulative reward of all actions performed by the agent in the state space, then

$$R\left(t\right) = \sum_{i=0}^{n} \gamma^i r_{t+i} \tag{24}$$

where γ denotes the discount rate, $\gamma \in \left[0,1\right]$.

Since minimizing the entire cost of terminal task processing is essentially the same as maximizing cost savings compared to local processing, it can be used as a reward function to solve the optimization objective.

CEC-A3C Algorithm

The CEC-A3C algorithm proposed in this paper consists of two deep neural networks: a strategy network (actor network) and a value network (commentator network). The strategy network controls the agent to perform a more efficient task unloading strategy $\pi(a_t \mid s_t)$ through neural network fitting. The value network is used to evaluate the state of the agent after performing the action, but it does not control the agent. The local network located in different agents interacts with its environment, calculates the gradient of the neural network loss function in its own environment, and periodically uploads it to the global network located in the cloud. The global network returns the updated parameters to the local network through learning to better guide the agents to perform actions, thus obtaining the optimal unloading strategy that meet optimization objectives.

The strategy network controls the agent to execute a better task unloading strategy through neural network fitting, with the aim of obtaining the maximum expected value of the discounted cumulative reward. Going by the ideas in this paper, when the strategy network makes the unloading decision, it needs to determine two factors: the number of channels allocated to the user subtask, and the quantity of computing resources allocated to the user subtask. When the number of subtasks in the system is large, it will lead to problems such as excessive dimension of action space and slow training of network parameters. Therefore, this paper advocates the use of the probability distribution strategy network since it can output multiple subactions at the same time to replace the traditional single-action output network. The use of the policy to optimize the network approximation policy function is defined as

$$\pi(a_t^\alpha, a_t^\beta \mid s_t; \theta) \approx \pi(a_t^\alpha, a_t^\beta \mid s_t) \tag{25}$$

where θ is the weight parameter in the policy network, a_t^α is the channel bandwidth allocated by the subtask, and a_t^β is the number of computing resources allocated by the subtask.

If $Q_\pi\left(s_t, a_t\right)$ is defined as the action value function, indicating the evaluation of the action a_t performed in the current state s_t, then

$$Q_\pi\left(s_t, a_t\right) = E[R_t \mid S_t = s_t, A_t = a_t] \tag{26}$$

The action-value network based on the value function is also a function fitted by a neural network, and its network parameter is ω. The value network can be used to approximate the action-value function, and evaluate the pros and cons of the action, as follows:

$$Q\left(s_t, a_t; \omega\right) \approx Q\left(s_t, a_t\right) \tag{27}$$

If the value function $V_\pi\left(s_t\right)$ represents the current state $s\left(t\right)$ using the task offloading strategy π, then the expected value of the cumulative reward that the agent can obtain will be

$$V_\pi\left(s_t\right) = \sum_a \pi(a \mid s_t) Q_\pi\left(s_t, a\right) \tag{28}$$

Since the optimization goal of the policy function is to maximize the reward, the current state value function $V_\pi\left(s\right)$ is calculated using the gradient ascent method. The time series difference error is used to fit the network, and then the updated formula of the policy gradient is

$$\nabla_\theta V_\pi\left(s\right) = E_{a\sim\pi}[\nabla_\theta log\pi(a \mid s;\theta) Q_\pi\left(s, a\right)] \tag{29}$$

where $\nabla_\theta log\pi(a \mid s;\theta) Q_\pi\left(s, a\right)$ is an unbiased estimate of $\nabla_\theta V_\pi\left(s\right)$.

However, since the actions in this paper have reward values that are all positive numbers and the gradient values are all greater than zero, the probability of each action will be continuously improved with the gradient ascent algorithm. This will result in an increase in gradient variance and slow down its learning rate. Therefore, the state value function is taken as the baseline function, and we increase the baseline function $V_\pi\left(s\right)$ to reduce the variance and ensure its unbiasedness. This is shown in Equation 30:

$$\nabla_\theta V_\pi\left(s\right) = E_{a\sim\pi}\left[\nabla_\theta log\pi(a \mid s;\theta)\left(Q_\pi\left(s, a\right) - V_\pi\left(s\right)\right)\right] \tag{30}$$

where $\left(Q_\pi\left(s, a\right) - V_\pi\left(s\right)\right)$ is the advantage function, which indicates the pros and cons of the action performed by strategy π_t when the parameter is θ. By assuming that the events of each subaction executed according to strategy π_θ are independent of each other, then

$$\pi_\theta\left(a_t^\alpha, a_t^\beta \mid s_t\right) = \pi_\theta\left(a_t^\alpha \mid s_t\right) \cdot \pi_\theta\left(a_t^\beta \mid s_t\right) \tag{31}$$

The parameter gradient of the multiaction output policy network is expressed as

$$d\theta = \nabla_\theta \log\left[\pi_\theta\left(a_t^\alpha | s_t\right)\cdot \pi_\theta\left(a_t^\beta | s_t\right)\right]\left(Q_\pi\left(s,a\right) - V_\pi\left(s\right)\right) = \nabla_\theta\left[\log\pi_\theta\left(a_t^\alpha | s_t\right) + \log\pi_\theta\left(a_t^\beta | s_t\right)\right]\left(Q_\pi\left(s,a\right) - V_\pi\left(s\right)\right) \tag{32}$$

To simulate the historical experience of subtask selection and offloading strategy selection, this paper uses the single-step TD-Target method in the Q-learning algorithm in order to improve the efficiency of the reinforcement learning algorithm. Therefore, the TD Target is:

$$y_t = r_t + \gamma \cdot \max_a Q(s_{t+1}, a) \tag{33}$$

The loss function of the value network is:

$$L\left(\omega\right) = \left[V\left(s\left(t\right), a\left(t\right); \omega\left(t\right)\right) - y_t\right]^2 \tag{34}$$

Then, the parameter gradient of the value network can be expressed as:

$$\omega = \frac{\partial\left[V\left(s\left(t\right), a\left(t\right); \omega\left(t\right)\right) - y_t\right]^2}{\partial\omega} \tag{35}$$

The detailed execution process is shown in Algorithm 1. In the cloud-edge-end collaborative environment, each edge server at the edge layer is equipped with an actor-commentator network.

1. First, the end user divides the coarse-grained computing task into fine-grained computing tasks that cannot be subdivided, and it obtains the topological sequence of subtasks through the directed acyclic graph after subtask division.
2. Second, the signal receiver AP collects the end user's subtask demand, channel bandwidth, computing resources and other environmental information through the sensor equipment to form the system status s_t.
3. Finally, the edge server randomly selects a task requirement in the task set (Step 7), the actor network is responsible for making the unloading strategy and executing action a_t^α, a_t^β, and then the system enters the next state and obtains rewards (Steps 8 and 9). Repeat the process until it exceeds the set number of times or reaches the termination state (step 11), and then delete the task requirement from the task set (Step 12).

The task selection process must be repeated until the task set M' is empty, and the reviewer network will obtain the expected value of cumulative rewards. The reward value is optimized using Equation 30, and the TD target and loss function are calculated using Equation 33 and Equation 34. The parameter gradients of the strategy network and value network are calculated separately, and the gradients are uploaded to the global network of the cloud layer (Steps 17 and 18). The final operation is to update the global network parameters of the cloud layer and send the updated global network parameters back to the local network for the next round of training.

Algorithm 1 CEC-A3C Algorithm
Input:
Task set M'
The maximum number of iterations $Iter_{max}$;

Topological sequence of subtask set $G^*\left(V,E\right)$;
Maximum tolerance time T_{max};
Output: optimal task offloading policy set $\pi^*(a_t^\alpha, a_t^\beta \mid s_t)$
1. Initialize the global policy network parameters θ, global value network parameter ω, globally shared time step $t'=0$
2. Initialize the local policy network parameters θ' in each edge server, local value network parameters ω', edge server time step $t=1$
3. For each edge node do
4. Synchronize parameters in the edge node $\theta'=\theta$, $\omega'=\omega$;
5. Set $t_0=t$;
6. Random select $m \in M'$
7. Repeat
8. select action a_t^α, a_t^β according to $\pi(a_t^\alpha, a_t^\beta \mid s_t;\theta')$
9. record the reward r_t and the new state s_{t+1} obtained by executing the action a_t;
10. $t=t+1$; $t'=t'+1$;
11. Until s_t termination status or $t'-t_{start}=t_{max}^{'}$
12. Delete the m in M'
13. Until $M'=\varnothing$
14. For $t-1$ do t_{start}
15. Optimize reward value function by equation (30)
16. Calculate TD Target according to (33) and $L\left(\omega\right)$ according to (34)
17. Update strategy accumulate gradient $d\theta$ according to:

$$d\theta = d\theta + \nabla_\theta\left[\log\pi_\theta\left(a_t^\alpha \mid s_t\right)+\log\pi_\theta\left(a_t^\beta \mid s_t\right)\right]\left(Q_\pi\left(s,a\right)-V_\pi\left(s\right)\right);$$

18. Update value accumulate gradient $d\omega$ according to:

$$d\omega = d\omega + \frac{\partial\left[V\left(s\left(t\right),a\left(t\right);\omega\left(t\right)\right)-y_t\right]^2}{\partial\omega};$$

19. End For
20. Update the gradient parameters of the global network

$$\theta=\theta-\rho_1 d\theta, \omega=\omega-\rho_1 d\omega$$

21. End For
22. Obtain the optimal set of offloading policy $\pi^*(a_t^\alpha, a_t^\beta \mid s_t)$
23. Obtain the optimal task sequence and task offloading strategy
24. End

EXPERIMENT RESULTS AND ANALYSIS

This section first gives a brief description of the experimental platform. To assess the practicability and efficacy of the CEC-A3C method put forward in this article, we conducted a series of simulation experiments for different system parameters in addition to comparing and analyzing the existing three task offloading methods.

Experimental Design

Using the programming language Python, we simulated data to validate the practicality of the proposed task offloading methodology. The task offloading overhead for end-users is assessed according to the usefulness of the multitask system. It adopts the cooperative scheme of distributed cloud server and edge server under an optical fibre and wireless hybrid network. The edge layer scenario is first set up, with five edge nodes, each covering a 5mx5m area. $\boldsymbol{M} = \left[4 \sim 6\right]$ smart devices are randomly distributed in this area, with the channel bandwidth at 5 MHz, and noise power at $\tilde{\boldsymbol{A}}^{2}_{\boldsymbol{m},\boldsymbol{p}} = 50\boldsymbol{dBm}$. By default, each terminal task has eight subtasks, one start subtask, and one end subtask. The data size of each subtask is randomly generated between [400,600] KB. The CPU frequency of the cloud server is 20 GHz, 10 GHz for the edge server, and 1 GHz for mobile users. For the purpose of this experiment, the energy expenditure coefficient of terminal equipment is set at $\boldsymbol{k} = 10^{-11}$, the energy expenditure coefficient of the edge server is $_{,\ p} = 10^{-9}$ cycle/J, and the time weight factor is α=0.5.

Experiment Analysis

In the CEC-A3C network, the learning rate of the network and the discount rate of the reward value are two important parameters that affect the convergence of the network. We set the learning rate of the CEC-A3C network to 0.005, 0.015 and 0.05 in order to train the CEC-A3C network, and the results are shown in Figure 5. Since in the CEC-A3C network, each local network in the edge layer performs network training asynchronously and in parallel, and the actor-critic networks in different environments are trained at different rates, the reward values obtained by the whole distributed edge computing system vary at a large rate at the beginning and then gradually stabilize. When the learning rate is 0.005, the iterative optimization of the CEC-A3C algorithm is less efficient and the reward values converge slowly. When the learning rate is 0.015, the CEC-A3C algorithm converges quickly and obtains a high cumulative reward. When the learning rate is 0.05, the algorithm reward value may cross the optimal value and fail to converge to the global optimum as the number of iterations increases due to the wide search range.

Figure 6 demonstrates the impact of the discount rate γ on the performance of the CEC-A3C algorithm. We set the learning rate to a fixed value of 0.015 and γ takes values of 0.3, 0.6, and 0.9 to train the CEC-A3C network. From the figure, it can be seen that when the discount rate is 0.3, the learning efficiency is poor, convergence is slow, and the reward value is low. When the discount rate is 0.6, the convergence speed is faster and the convergence value increases significantly. When the discount rate is 0.9, the algorithm achieves better performance and the final moving average reward converges to approximately 520. A larger the discount rate results in faster convergence of the reward value and higher convergence value. The discount rate indicates whether the expected future rewards or the expected rewards in the present are considered in the learning process of the CEC-A3C network. Since this paper needs to consider more task offloading in the future, after conducting several simulations with different discount rates, we choose a network discount rate of 0.9.

To verify the performance of the dominance function of the algorithm, we set the learning rate size at 0.015 and the discount rate size at 0.9. Figure 7 shows the convergence performance of the dominance function of the cloud global network after receiving feedback from the edge nodes. The figure shows that the dominance function can always converge to stability after approximately 500 iterations. Therefore, the cloud global network model, whenever receiving new edge node parameter

Figure 5. Reward values obtained by the network at different learning rates

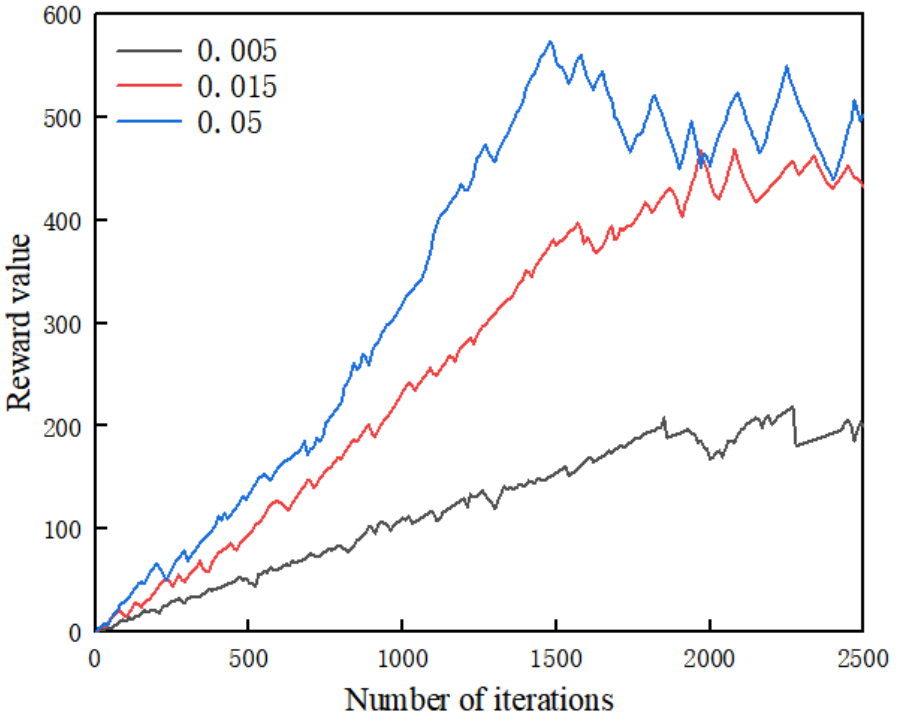
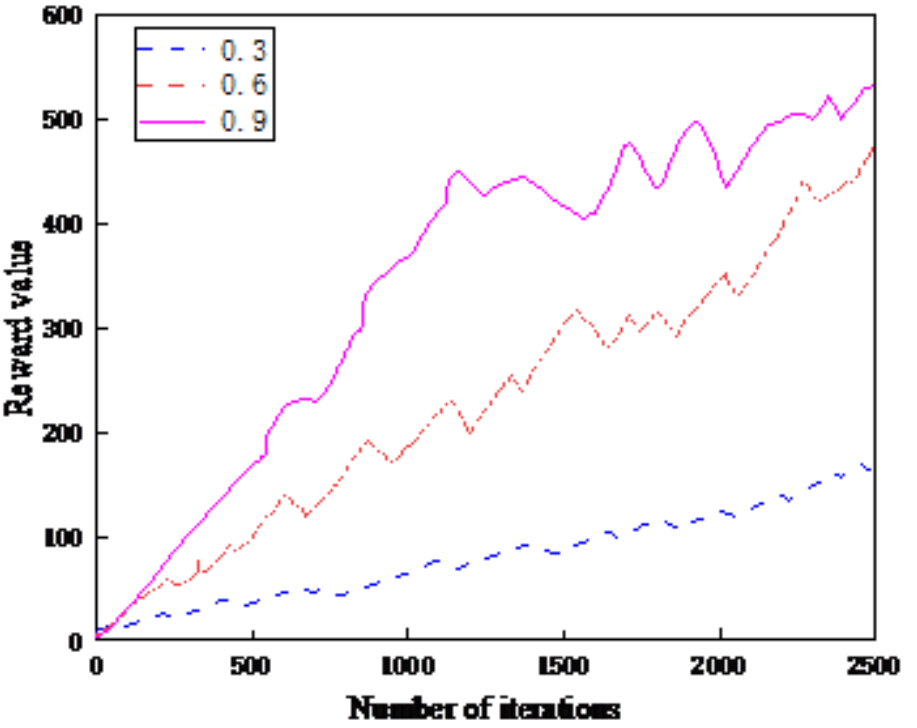


Figure 6. Value of rewards obtained by the network at different discount rates



gradients, can reach the convergence of the dominance function within a finite number of iterations so that the ideal network parameters can be learned.

Figure 8 shows the convergence efficiency of the loss function L(ω) in the algorithm of this paper. We set the learning rate size at 0.015 and the discount rate size at 0.9. From the figure, it can be seen that the data of the loss function vary greatly at the beginning of training, and then the value of L(ω) gradually decreases after approximately 1,000 iterations. This is mainly because during the initial training, the network is in the update phase and the actions performed have a large impact on the reward value, so the value of the loss function changes drastically. As the number of iterations increases during the training process, the value of the loss function gradually decreases and then stabilizes after approximately 1,600 iterations. Therefore, the algorithm in this paper can automatically update the unloading decision, converge to the optimal value, and finally learn the optimal network parameters.

We choose to compare and analyse the CEC-A3C algorithm with all subtask local processing (FLP), subtask full offload processing (FOP), and the genetic-simulated annealing algorithm (GSA-EDGE) (Wang et al., 2022) to demonstrate the advanced nature of the CEC-A3C algorithm. Considering that the terminal subtasks in the GSA-EDGE scheme can be offloaded to the cloud for processing, we modify the GSA-EDGE algorithm according to the system model in this paper. Figure 9 presents the total system cost for different algorithms under different average numbers of subtasks. It is evident that there is a gradual increase in the total cost of system task processing for all methods as the average number of subtasks increases. Among them, the total consumption of system task processing is higher than that of the GSA-EDGE and CEC-A3C schemes because there is no task offloading strategy for the FLP and FOP schemes. Also, due to the limited computing resources of the terminal equipment, the cost of FLP is the highest. Compared with GSA-EDGE, the sum consumption

Figure 7. Convergence process of the dominance function

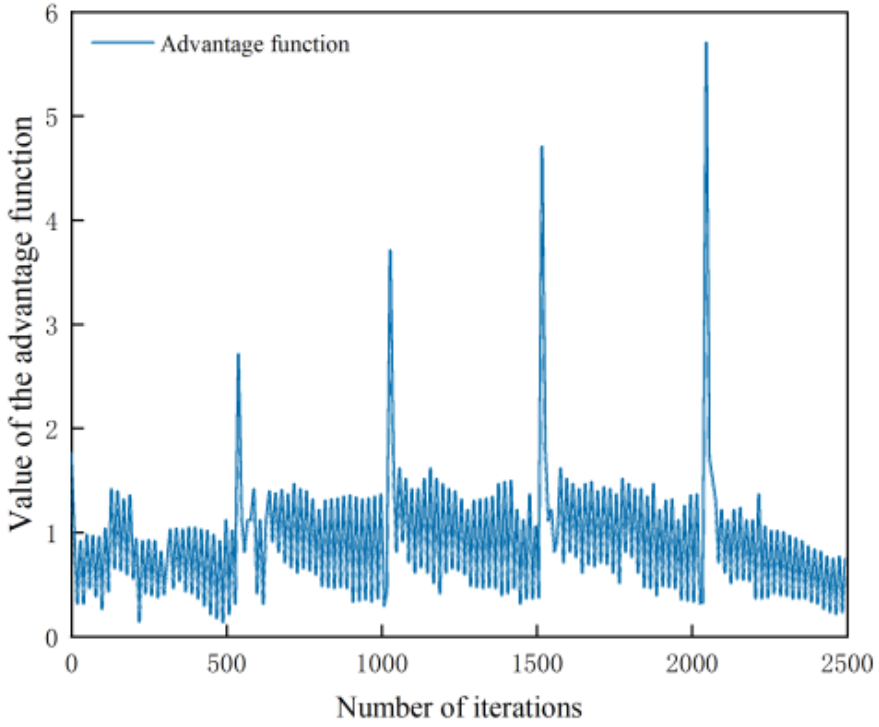


Figure 8. Convergence process of loss function

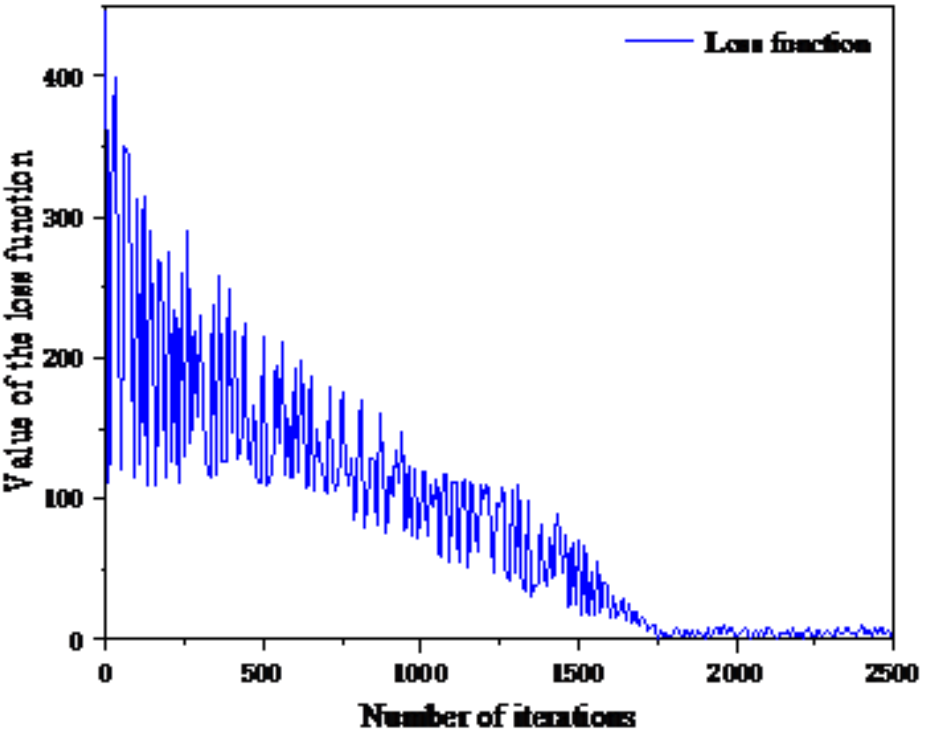
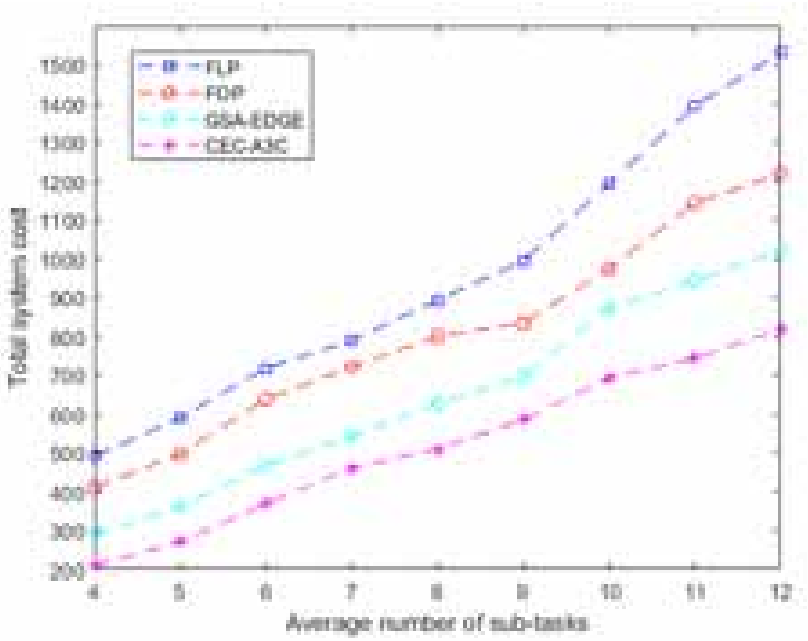


Figure 9. System cost for different average numbers of subtasks



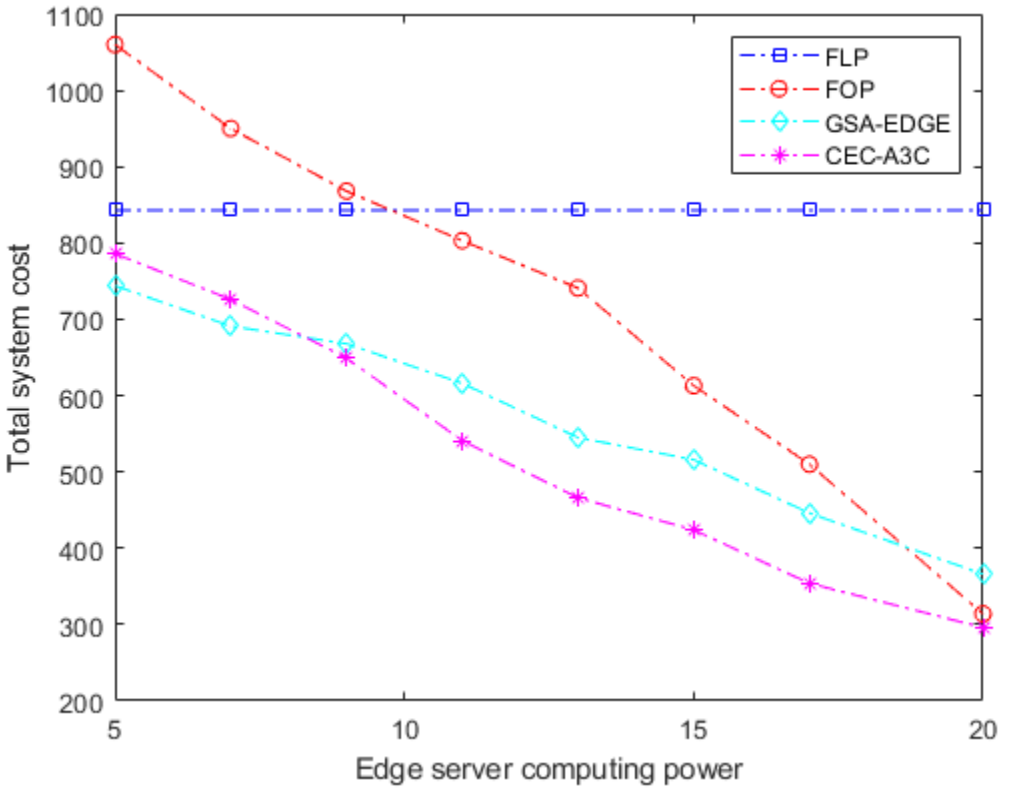
of the CEC-A3C algorithm is lower, and as the quantity of subtasks increases, the discrepancy in cost between the two schemes grows. Therefore, the CEC-A3C scheme can decrease the processing consumption of fine-grained tasks to a limited extent, and the effect is more pronounced when the average number of subtasks is large.

Figure 10 shows the total system cost of different algorithms under different edge server resources. Since the system cost in the FLP scheme only depends on the computing power of the local device and has nothing to do with the processing power of the edge node, the cost is not affected. As the computing resources of the edge node increase, the processing time of subtasks offloaded to the edge node decreases, resulting in a gradual reduction in the system cost. The FOP scheme is highly dependent on edge servers and has the largest delay change rate. When the computational resources of the edge node are lower than 8 GHz, the total cost of the CEC-A3C algorithm is significantly lower than that of the GSA-EDGE algorithm and decreases steadily. Therefore, by increasing the computational resources of the edge node, the CEC-A3C algorithm is capable of significantly reducing both system processing delay and energy utilization.

CONCLUSION

To address the task offloading problem for subtask-dependent models in partial offloading models of edge computing, this paper constructs a fine-grained partitioning model of terminal tasks in the cloud-edge architecture. The optimization goal is to reduce the sum of minimized task offloading delay and energy consumption weight. In order to achieve this aim, this paper proposes an

Figure 10. System cost of different edge server computing capabilities



asynchronous dominant actor-critic algorithm under cloud-edge collaboration. The algorithm captures multidimensional data from the environment and performs asynchronous online learning at multiple edge nodes, thus realizing online optimal offloading decisions for end-tasks, reducing the total cost of the end-task offloading system, and improving the user experience. Finally, the efficiency of the CEC-A3C algorithm is verified through simulation experiments. In future research, edge computing technology will be combined with federated learning technology to further enhance the protection of user privacy in task offloading.

ACKNOWLEDGEMENTS

This paper was supported by the Natural Science Foundation Project of Hebei Province,China(No. F2021207005)

REFERENCES

Alam, M. G. R., Hassan, M. M., Uddin, M. Z., Almogren, A., & Fortino, G. (2018). Autonomic computation offloading in mobile edge for IoT applications. *Future Generation Computer Systems*, *90*(JAN), 149–157.

Chae, H., You, C., & Kim, B. (2017). Energy-efficient resource allocation for mobile-edge computation offloading. *IEEE Transactions on Wireless Communications*, *16*(3), 1397–1411. doi:10.1109/TWC.2016.2633522

Chen, L., Wu, J., Dai, H., & Huang, X. (2018). BRAINS: Joint bandwidth-relay allocation in multihoming cooperative D2D networks. *IEEE Transactions on Vehicular Technology*, *67*(6), 5387–5398. doi:10.1109/TVT.2018.2799970

Chen, M., & Hao, Y. (2018). Task offloading for mobile edge computing in software defined ultra-dense network. *IEEE Journal on Selected Areas in Communications*, *36*(3), 587–597. doi:10.1109/JSAC.2018.2815360

Chen, X., Jiao, L., Li, W., & Fu, X. (2016). Efficient multi-user computation offloading for mobile-edge cloud computing. *IEEE/ACM Transactions on Networking*, *24*(5), 2795–2808. doi:10.1109/TNET.2015.2487344

Dong, P., Wang, X. X., Rodrigues, J. J. P. C., & Ning, Z. (2019). Deep reinforcement learning for vehicular edge computing: An intelligent offloading system. *ACM Transactions on Intelligent Systems and Technology*, *10*(6), 1–24. doi:10.1145/3531230

Eshraghi, N., & Liang, B. (2019). Joint offloading decision and resource allocation with uncertain task computing requirement. *IEEE Infocom 2019 - IEEE Conference on Computer Communications*, 1414-1422. doi:10.1109/INFOCOM.2019.8737559

Khan, W. Z., Ahmed, E., Hakak, S., Yaqoob, I., & Ahmed, A. (2019). Edge computing: A survey. *Future Generation Computer Systems*, *97*, 219–235. doi:10.1016/j.future.2019.02.050

Li, J., Zhang, H., Ji, H., & Li, X. (2019). Joint computation offloading and service caching for MEC in multi-access networks. *2019 IEEE 30th Annual International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC).* doi:10.1109/PIMRC.2019.8904110

Liu, Y., Yu, H., Xie, S., & Zhang, Y. (2019). Deep reinforcement learning for offloading and resource allocation in vehicle edge computing and networks. *IEEE Transactions on Vehicular Technology*, *68*(99), 1–1. doi:10.1109/TVT.2019.2935450

Mao, Y., You, C., Zhang, J., Huang, K., & Letaief, K. B. (2017). A survey on mobile edge computing: The communication perspective. *IEEE Communications Surveys and Tutorials*, *19*(4), 2322–2358. doi:10.1109/COMST.2017.2745201

Mao, Y., Zhang, J., Song, S. H., & Letaief, K. B. (2017). Stochastic joint radio and computational resource management for multi-user mobile-edge computing systems. *IEEE Transactions on Wireless Communications*, *16*(9), 5994–6009. doi:10.1109/TWC.2017.2717986

Poularakis, K., Llorca, J., Tulino, A. M., Taylor, I., & Tassiulas, L. (2019). Joint service placement and request routing in multi-cell mobile edge computing networks. *IEEE INFOCOM 2019 - IEEE Conference on Computer Communications,* 10-18. doi:10.1109/INFOCOM.2019.8737385

Shakarami, A., Ghobaei-Arani, M., & Shahidinejad, A. (2020). A survey on the computation offloading approaches in mobile edge computing: A machine learning-based perspective. *Computer Networks*, *182*(9), 107496. Advance online publication. doi:10.1016/j.comnet.2020.107496

Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, *3*(5), 637–646. doi:10.1109/JIOT.2016.2579198

Sun, Q. B., Liu, J., Li, S., Fan, C. X., & Sun, J. J. (2010). Internet of things: Summarize on concepts, architecture and key technology problem. *Journal of Beijing University of Posts and Telecommunications*, *33*(3), 1–9.

Tang, M., & Wong, V. W. S. (2022). Deep reinforcement learning for task offloading in mobile edge computing systems. *IEEE Transactions on Mobile Computing*, *21*(6), 1985–1997. doi:10.1109/TMC.2020.3036871

Tran, T. X., Hajisami, A., Pandey, P., & Pompili, D. (2017). Collaborative mobile edge computing in 5G networks: New paradigms, scenarios, and challenges. *IEEE Communications Magazine*, *55*(4), 54–61. doi:10.1109/MCOM.2017.1600863

Tran, T. X., & Pompili, D. (2017). Joint task offloading and resource allocation for multi-server mobile-edge computing networks. *IEEE Transactions on Vehicular Technology*, *68*(1), 856–868. doi:10.1109/TVT.2018.2881191

Wang, S., Wang, W., Jia, Z., & Pang, C. (2022). Flexible task scheduling based on edge computing and cloud collaboration. *Computer Systems Science and Engineering*, *42*(3), 1241–1255. doi:10.32604/csse.2022.024021

Warren, J. (2015). *Big data: Principles and best practices of scalable realtime data dystems. In Big Data: Principles and Best Practices of Scalable Realtime Data Systems*. Simon and Schuster.

Zhan, Y., Guo, S., Li, P., & Zhang, J. (2020). A deep reinforcement learning based offloading game in edge computing. *IEEE Transactions on Computers*, *69*(6), 883–893. doi:10.1109/TC.2020.2969148

Zhao, N., Liang, Y. C., Niyato, D., Pei, Y., & Jiang, Y. (2019). Deep reinforcement learning for user association and resource allocation in heterogeneous cellular networks. *IEEE Transactions on Wireless Communications*, *18*(11), 5141–5152. doi:10.1109/TWC.2019.2933417

Zhao, Z., Liu, F., Cai, Z., & Xiao, N. (2018). Edge computing: Platforms, applications and challenges. *Journal of Computer Research and Development*, *55*(002), 327–337. doi:10.7544/issn1000-1239.2018.20170228

Zheng, J., Gao, L., Wang, H., Li, X., & Yang, X. (2019). Joint downlink and uplink edge computing offloading in ultra-dense HetNets. *Mobile Networks and Applications*, *24*(5), 1452–1460. doi:10.1007/s11036-019-01274-y

Suzhen Wang, born in 1964, Ph.D., professor. Her research interests include mobile cloud computing, edge computing, big data processing and analysis.

Yongchen Deng, born in 1996, M.S. candidate. His research interests include edge computing and cloud coordination.

Zhongbo Hu, born in 1998, M. S. His research interests include edge computing and cloud coordination.