Edge Intelligence: A Computational Task Offloading Scheme for Dependent IoT Application

Han Xiao, Changqiao Xu, *Senior Member, IEEE,* Yunxiao Ma, Shujie Yang, Lujie Zhong, and Gabriel-Miro Muntean, *Senior Member, IEEE*

***Abstract*—Computational offloading, as an effective way to extend the capability of resource-limited edge devices in Internet of Things (IoT), is considered as a promising emerging paradigm for coping with delay-sensitive services. However, on one hand, applications commonly include several subtasks with dependent relations and on the other hand, the dynamic changes in network environments make offloading decision-making become a coupling and complex NP-hard problem, difficult to address. This paper proposes an intelligent Computational Offloading scheme for Dependent IoT Application (CODIA), which decouples the performance enhancement problem into two processes: scheduling and offloading. First, a prioritized scheduling strategy is designed and its complexity is analyzed. Then, an offloading algorithm with offline training and online deployment is introduced. Due to the temporal continuity between subtasks, the dependency relation is transformed into a transition of device state, and the overhead for the whole application is considered to be the long-term benefit. CODIA leverages an Actor-Critic-based solution, where the IoT devices are able to deploy intelligent models and dynamically adjust the offloading strategy to achieve low latency, while controlling energy consumption. Finally, a series of experiments are conducted to verify the robustness and efficiency of the proposed solution in terms of convergence, latency, and energy consumption.**

***Index Terms*—Edge Intelligence, Computational Offloading, Dependent Application, Deep Reinforcement Learning**

The latest rapid development of the Internet of Things (IoT) makes possible interconnection of increasing number of devices (e.g. Cisco estimates there will be about 75.4 billion interconnected devices by 2025[1]). These devices will support the next generation of applications including virtual reality-based (VR)[2], autonomous driving[3], etc. Compared with traditional services that only require a few computational steps, the emerging applications have much higher demands in terms computing resources [1]. The level of support for these computational requirements closely influences the associated quality of experience (QoE) of users. However, following the latest invalidation of Moore's Law[4] and the exponential increase in computing resource demands of emerging applications, it is no longer feasible to expect that all resource requests are addressed locally in time. In this context, providing support for users with resource-limited IoT devices in order to have access to high QoE services is a challenge that must be addressed.

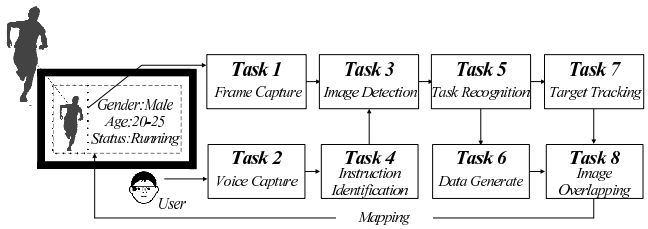


Fig. 1. The dependency task graph of a recognition application

Edge computing (EC)[5] is regarded as a promising emerging paradigm to solve the IoT problem mentioned above [3]–[6]. According to a report from App Annie, the average usage time of edge IoT nodes (e.g. phone, pad, etc.) is only 4.2 hours/day [2], so their associated computing resources are available most of the time. This phenomenon provides available condition for EC. By offloading computational tasks at the edge, close to the task generator (i.e. requester), the results can be returned to the requester quickly and the overload of the core network can be avoided. This also enables support for various other IoT scenarios such as wearable computing [8], industrial-level applications [9], and the Internet of vehicles [10]. However, there are still several challenges which reduce the effectiveness of deployed computational offloading strategies in real-world environments. They motivate continuing the work to propose innovative new solutions to address these challenges and are summarized as follows:

1) The IoT devices at the edge are generally battery-powered and need to control their energy consumption, while still providing high QoE to users [11], [12]. Although computational offloading expands the available resources, it also increases the pressure put on devices [13]. Therefore, these IoT devices have to balance between the QoE (e.g. latency) and their energy consumption. However, there are different understandings of this balance for different IoT applications [13]–

Manuscript received April 5, 2021; revised August 23, 2021; revised January 18, 2021; accepted March 2, 2022. This work is supported by the National Natural Science Foundation of China (NSFC) under grant No. 61871048, 61872253, and 62001057 and by the 111 Project (B18008). G.-M. Muntean wishes to acknowledge the Science Foundation Ireland (SFI)'s support via grant nos. 16/SP/3804 (Enable) and 12/RC/2289_P2 (Insight). (*Corresponding author: Changqiao Xu.*)

H. Xiao, C. Xu, Y. Ma and S. Yang are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100876, P.R. China. E-mail: {xiaohan, cqxu, myx, sjyang}@bupt.edu.cn

L. Zhong is with the Information Engineering College, Capital Normal University, Beijing 100048, China. E-mail: zhonglj@cnu.edu.cn.

G.-M. Muntean is with the Performance Engineering Laboratory, School of Electronic Engineering, Dublin City University, Dublin 9, Ireland. E-mail: gabriel.muntean@dcu.ie.

[1]https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html

[2]https://arvr.google.com/

[3]https://www.grandviewresearch.com/industry-analysis/autonomous-vehicles-market

[4]https://www.investopedia.com/terms/m/mooreslaw.asp

[5]https://en.wikipedia.org/wiki/Edge_computing

[15], and therefore there is a strong requirement in terms of proposing flexible energy-quality control strategies.

2) A typical IoT application (e.g. the recognition application of Google glasses[6], etc.) is composed of a set of subtasks with complex dependencies [7], [17]–[22], as shown in Fig. 1. Offloading the application as a whole simplifies the scheduling decision, which is the premise of most research efforts [8]–[15]. However, the parallelism within the application is not being fully utilized in this way [19], [23]. At the same time, current dependent task offloading schemes are mostly applied in specific scenarios and difficult to meet the requirements of diversified computing applications [16], [19], [25]. There is a need for a dynamic approach for offloading that can be applied to a large range of IoT applications with dependencies.
3) The network environment presents complex and dynamic characteristics due to the time-varying wireless communication conditions and highly variable user requests [26], [27]. On one hand, optimal decisions in an IoT scenario are not always beneficial to long-term scheduling [20], [23]. On the other hand, even in similar conditions, it is not unusual for the computational tasks to make different offloading choices when faced with the same alternatives. These indicate the need for designing solid performance-oriented offloading solutions.

These challenges make computational offloading complex and finding an efficient and flexible solution is non-trivial. Under these circumstances, the progress of artificial intelligence (AI) technology in recent years contributed with new ideas in the quest to address the challenges mentioned above [28]. By training an intelligent model (e.g. deep neural network) on a platform with rich computing resources [29], it can help the decision-maker to identify the best choice according to the current state. This gives it the ability to respond to various environmental challenges with a mature strategy. Noteworthy is that the EC paradigm provides support for AI. Today's mobile chips are endowed with considerable computing power (comparable to computing servers of a decade ago [30]) and have the potential to execute complicated models at the edge. In this context, this paper proposes an innovative solution with edge intelligence for Computational tasks Offloading for Dependent IoT Applications (CODIA). A series of experimental tests are carried out to verify the effectiveness of the proposed CODIA solution in terms of convergence, energy consumption and latency. The experimental results show that CODIA reduces the latency by 25% compared with other schemes, while maintaining very good energy efficiency.

The **main contributions** of this paper are as follows.

1) We introduce a performance-aware application-level offloading model and adopt a general directed acyclic graph to establish parallelism for dependent subtasks. We transform the formulated overhead minimization problem into a traveling salesman problem and prove its property of NP-hardness through a rigorous theoretical proof. The dependencies are clearly considered in this model, unlike in the sequential subtask model, which alleviates the coupling of dependencies and improves the universality of proposed solution.
2) We design a generator-executor Multi-Queue Priority scheduling algorithm (MQP) focused on task finishing time. This algorithm transforms successive offloading processes into queue updates to decrease scheduling latency and achieves this with reduced overhead and low complexity.
3) We propose an intelligent offloading algorithm for offline learning and online deployment, which adopts the deep reinforcement learning framework based on an *Actor-Critic* approach. A penalty term for timeout is added to the reward function and the Monte-Carlo method is applied to estimate the obtainable benefit and improve the adaptability to the dynamic environment. This algorithm makes our solution more responsive to diverse application requirements and is applicable to complex network environments.
4) We design a complete experimental environment and evaluate the offloading efficiency of the proposed scheme for application-level tasks through a series of experiments, including convergence verification, overhead analysis with the specific component, etc. *CODIA* outperforms other four state-of-the-art schemes in terms of delay and overhead.

The rest of this paper is organized as follows. Section II discusses the related works in the field published in the recent years. In Section III, the system model is designed and the minimization problem is formalized. The dependent subtask scheduling scheme is presented in Section IV. A detailed description of the offloading scheme is provided in Section V. The experimental testing is described and its results are analyzed in Section VI. Finally, Section VII concludes this paper.

I. Related Works

In recent years, the goal of computational offloading is gradually changing from the whole application to the level of atomic subtasks. This section provides a brief overview of the related works on computational offloading and highlights the differences between this article and previous research.

A. Edge Computational Offloading

Benefiting from the improvement in computing capabilities of edge devices, edge computing emerged in the second decade of the current century as a key paradigm for addressing the performance limitations of cloud computing. It involves provision of computational support by means of offloading. Up to now, edge computing offloading has been divided into two main categories. The first category includes application-level (full) offloading (i.e. binary offloading) [32]–[36], where the entire application is transferred to another powerful node for processing. For example, Chen *et al.* [32] deployed a three-tier mobile computing framework involving the cloud, edge nodes, and users. Through vertical cooperation between

[6]https://www.google.com/glass/start/

hierarchical nodes, computing overhead was significantly reduced. Our research group has employed an idea from the game theory to optimize the multimedia service supported by edge computing, in order to realize efficient services and guarantee the reliability of transmission [34], [35]. We have also designed an augmented graph model to realize a joint optimization of computation and transmission [36].

The second category is partial offloading [37], [38], involves offloading a part of the original task by splitting the application. For example, Liu *et al.* [37] proposed a price-based method to manage computational resources, where the computation data is assumed to be arbitrarily divided bit-wise. Similarly, You *et al.* [38] consider the data can be split for separate computing in the context of resource allocation in a multiuser edge offloading system. In many works, the pattern of the proposed partial offloading is idealistic because the segmentation of tasks can hardly be regarded as continuous. For example, the driverless application-level task in the Internet of vehicles. The terminal needs to perceive the road condition first, then plan the driving path according to the information, and finally adjust the speed of the vehicle. It is difficult to offload partial task data to another device, which violates the execution rules of computer program. Therefore, researchers proposed to represent application tasks by special models and use those in the offloading process.

B. Dependent Application Tasks

At present, researchers subdivide the application tasks into atomic subtasks with different responsibilities according to requirements [17]–[25]. However, many computing subtasks are dependent on each other, like the example of driverless mentioned above. Before a precursor of a dependent subtask completes, the subsequent task is locked because it has not received the input data it requires to complete. This makes offloading difficult and diverse scholars have considered using discrete task description models to represent applications [24], [25]. For example, Mehrabi *et al.* [24] designed a three-node MEC system to minimize the energy consumption during offloading and considered the application task as a group of sequentially dependent tasks. Kao *et al.* [25] treated a task as a serial trees and proposed an online offloading algorithm to reduce processing latency.

Actually, the proposed serial sequential task model is only applicable to tasks in some special scenarios and unfortunately it cannot be applied to broader application service requirements. Therefore, a more general graph structure task model needs to be considered. In [19], Han *et al.* proposed a heuristic algorithm to solve the task offloading by employing a graph structure and minimizing the overhead in an ultra-dense edge network. Liu *et al.* [22] developed an efficient task scheduling algorithm to guarantee the completion time constraints of graphic applications. And it mainly considers roadside units as the computing execution node and ignores other available devices, such as adjacent vehicles, which makes the idle computing resources underutilized. And unfortunately even the most representative solutions discussed (i.e. heuristic search [19] and convex relaxation [23]) cannot guarantee

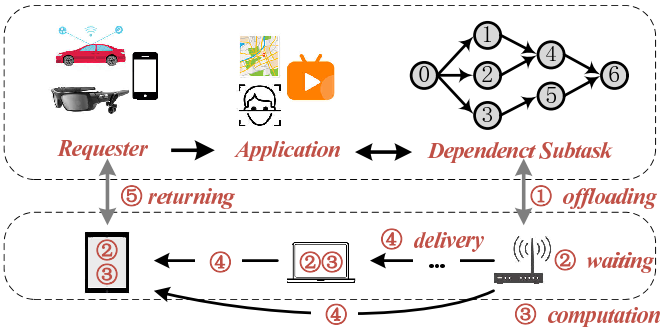


Fig. 2. Computational Communication Scenario

superior performance as they can easily be limited to a local optimum solution. Additionally, the dynamic time-varying wireless environment and frequent computations caused by the various application service requirements make them less practical.

C. Artificial Intelligence-driven Offloading

Driven by the latest advancements in neural networks and not only, many Artificial intelligence (AI)-based solutions have been proposed lately [28]–[30]. As an efficient function approximator, a deep neural network can map high-dimensional and multi-scale states to actions by expanding the number of neurons or network layers. At the same time, reinforcement learning provides a basis for action selection through state-action values, thus improving the decision-making efficiency. These aspects make AI-based schemes have unique advantages in dealing with a dynamic environment and diversified task requirements, and have attracted increasing research interest. For example, Qiu *et al.* [39] adopted a blockchain-empowered mobile edge computing approach based on reinforcement learning to accommodate highly dynamic environments and address the large computational complexity. Qi *et al.* [40] formulated the offloading decision as a long-term planning problem and employed the deep reinforcement learning to obtain the optimal solution. Tang *et al.* [41] incorporated the LSTM structure and a deep Q-network to reduce the ratio of dropped tasks and average delay. However, the above work is mainly used to solve application-level task offloading and to date there is a lack of effective solutions for dependent tasks offloading.

Unlike the existing work mentioned above, this paper combines queue control-empowered task scheduling and AI-based offloading scheme to improve the performance of edge-enhanced IoT services for task dependent applications in highly dynamic network environments.

II. System Model

Fig. 2 illustrates a computational communication scenario in IoT, considered in this paper. The edge IoT device (e.g. smart car, google glasses, mobile phone, etc.) plays the role of the requester, generating computational tasks based on the applications(e.g. navigation, multimedia, recognition, etc.) invoked by the user. The application consists of a set of dependent subtasks. Thus, computational offloading is defined as sequential migrations of subtasks. The subtasks are successively transmitted to a set of edge nodes(e.g. laptop, pad,

base station with edge server, etc.) and wait for execution. We assume that, by means of an incentive mechanism such as [42], the idle nodes are motivated to participate in computing cooperation. Due to the dependent relations, the transitivity of the transmission between executors is presented. Finally, after a series of computations and deliveries, the result is returned to the requester. Next, the system model is introduced. It includes the following three components: network model, task model, and overhead model.

Note that we use calligraphy symbol as set or graph, e.g. $\mathcal{G}, \mathcal{N}$. The **bold** type corresponding vector or function. The regular symbols indicate element or variable, e.g. c_m, P_i. And the space is denoted by blackboard bold, e.g. $\mathbb{S}$, $\mathbb{R}$. The size of set or vector is expressed by norm, e.g. $|Q_i^w|$. More mathematical notations are listed in Table I.

TABLE I
MATHEMATICAL NOTATIONS

Symbol	Description
$\mathcal{N}$	The communication element set
$\mathcal{G}$	The directed graph for the application invoked by user
h, m, n	Symbols for different dependent subtasks
i, j, k	Symbols for different communication nodes
$Q_i^w, \lvert Q_i^w \rvert$	Subtask waiting queue and its length
$\mathcal{T}, \mathcal{T}_m$	Subtask set and a dependent subtask m
τ	The deadline of the whole application
$\mathcal{E}, \mathcal{E}_{m,n}$	Edge set and a element of the dependency relations
$\mathbf{p}(\cdot), \mathbf{s}(\cdot)$	Precursor function and the successor function
$\mathrm{SIR}_{i,j}$	The signal-to-interference ratio between node i and j
d_i^m	Offloading decision for subtask m on node i
$Z_{i,j}$	The interference in the same channel for node i
κ_i	The constantly architecture parameter for node i
$O_i^{\mathcal{G}}$	The overhead of the application $\mathcal{G}$
$t_i^{ls,m}$	Latest start time of subtask m
$\mathbb{S}, \mathbb{A}, \mathbb{R}$	The space of state, action and reward
$\mathcal{S}_t, \mathcal{A}_t, \mathcal{R}_t$	The state, action and reward in slot t
s, a	A specific state and action
$\varphi(\varphi'), \vartheta(\vartheta')$	Actor and Critic(current network and target network)
$\mu(\mu'), \theta(\theta')$	The parameter of Actor and Critic
Q_i^p	Subtasks priority queue for the subtasks
Q_f	Latest finished subtasks queue
Q_e	Current schedulable subtasks queue

A. Network Model

The various devices in the communication system is represented by the node set $\mathcal{N} = \{1, ..., i, ..., N\}$. And node i, located at $(x_i, y_i), \forall i \in \mathcal{N}$, is equipped with a processor[7] that can perform the computational tasks. In order to guarantee the computation collaboration of edge nodes, the availability identifier ς_i is designed. $\varsigma_i = 0$ when the node is unavailable (e.g. out of communication range or in CPU-hungry working mode, etc.), and $\varsigma_i = 1$ otherwise. And the tasks arriving at node i are placed at the end of the task waiting queue Q_i^w. The first come, first served (FIFO) principle is adopted in this process. Additionally, wireless communication links are considered between the nodes. On one hand, cellular communications are established between devices and base station with edge server. On the other hand, device-to-device (D2D) communications based on a 5G protocol [43] are adopted between the edge IoT nodes. The edge server is considered to have rich computing resources so it can quickly complete the tasks offloaded from a requester.

B. Task Model

Assume a node (task generator) produces one application-level tasks at a time. The application consists of several dependent atomic subtasks, which can be offloaded to a group of adjacent edge nodes(task executor). In this paper, the application is represented as a directed acyclic graph (DAG) $\mathcal{G} = \{\mathcal{T}, \mathcal{E}, \tau\}$. $\mathcal{T} = \{\mathcal{T}_1, ..., \mathcal{T}_{|\mathcal{T}|}\}$ is the point set in $\mathcal{G}$, corresponding to the dependent subtasks. $\mathcal{E} = \{\mathcal{E}_1, ..., \mathcal{E}_{|\mathcal{E}|}\}$ is the edge set, which represents the dependency relation. Specifically, each dependency relation is denoted as a tuple (m, n), which is formalized as a directed line from subtask m to n. This indicates that the input of subtask n requires the output of task m, that is, n *depends* on m. In other words, m is the *precursor* of n and n is the *successor* of m. For clarity, we set up two functions $\mathbf{p}(\cdot)$ and $\mathbf{s}(\cdot)$ to represent the relation of *precursor* and *successor*, respectively (i.e. $m \in \mathbf{p}(n)$ and $n \in \mathbf{s}(m)$). It is worth noting that for complex applications, a subtask can depend on multiple precursors or can have multiple successors simultaneously. τ is the final deadline, which indicates that the application should be completed with this time constraint. For instance, in the case of entertainment applications, not meeting this deadline results in the decrease of service satisfaction, and in the case of industrial applications, overtime leads to economic loss and even life risk (i.e. driverless applications).

In order to provide a more general task model, two virtual nodes are added to DAG $\mathcal{G}$, identifying the start and end points of the application(e.g. the subtask 0 and 6 in Fig. 2), respectively. Therefore, the task graph can be identified as a structure from the start point to the end point. In this structure, each subtask is specifically represented as a tuple. For subtask m, $\mathcal{T}_m = \{c_m, q_m, p_m, \mathbf{d}_m\}$, where c_m is the required computing resources (e.g. CPU cycles), q_m is the input data size, p_m is the output data size and $\mathbf{d}_m = \{d_0^m, ..., d_N^m\}$ is the decision vector for offloading subtask m. Each element d_j^m can be expressed as follows:

$$d_j^m = \begin{cases} 1 & if\ subtask\ m\ is\ executed\ on\ node\ j \\ 0 & otherwise \end{cases}, \quad (1)$$

where $m \in [2, |\mathcal{T}| - 1], j \in [1, N]$. It is worth noting that i denotes the requester and task m is executed locally when $j = i$.

Note that $\mathcal{T}_0$ is the start point and has no precursor, whereas $\mathcal{T}_{|\mathcal{T}|}$ is the end point and has no successor. The computational resources required for both the start and end points are 0.

C. Overhead Model

As the subsequent subtasks depend on the execution of their predecessor subtasks, the overhead model is more complex than when a single task is processed. This subsection introduces the overhead model considering the following two aspects.

[7]This assumption can be easily extended to multiple processors with the FIFO principle. And the single processor is assumed here for readability.

1) Latency: From a macro perspective, the latency of executing a subtask mainly consists of transmission delay (including the offloading delay from the requester, the delivery delay between the edge nodes performing of dependent subtasks, and the delay associated with the return of results, etc.), computational delay, and waiting delay (i.e. the time consumed when the subtask waits for allocating computing resources in the task waiting queue).

First, the transmission delay mainly depends on the data size and transmission rate.For clarity, we denote the latency of offloading subtask m from requester i to node j as $t_{ij}^{o,m}$ and the latency associated with returning the execution result as $t_{ji}^{r,m}$. The latency of delivering the results of the precursor task h from the node k(the executor of h) to the node j (the executor of m) is denoted as $t_{kj}^{d,h}$. It is worth noting that when the node j is not reachable for the node k, it has to be delivered to the former with the requester i as the relay. So in this case, $t_{kj}^{d,h} = t_{ki}^{r,h} + t_{ij}^{o,h}$.

We assume the edge nodes adopt the Orthogonal Frequency Division Multiplexing (OFDM) [51] for accessing the base station. Different subcarriers are orthogonal and do not interfere with the others. However, the interference occurs to multiple adjacent BSs using the same subcarrier. The signal-to-interference-plus-noise ratio (SINR) [52] for the channel model with path loss and Rayleigh fading is:

$$\mathrm{SINR}_{i,j}^{w} = \frac{P_i h_{i,j} s_{i,j}^{-\alpha}}{Z_{i,j} + N_0}, \tag{2}$$

where P_i is the transmission power of sender i, $h_{i,j}$ is the channel gain following exponential distribution, i.e. $h_{i,j} \sim \exp(1)$ [3], $s_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$ is the distance between node i and j. The background noise is considered as the additive white Gaussian noise (AWGN) with the noise power N_0. $Z_{i,j}$ is the interference from the adjacent BS set Θ in the same channel (which can be measured at edge IoT nodes), which is expressed as:

$$Z_{i,j} = \sum_{k \in \Theta \setminus i} P_k h_{k,j} s_{k,j}^{-\alpha}, \tag{3}$$

where α is the path loss exponent.

Therefore, the communication rate of this process is:

$$r_{i,j}^{w} = w_i \log_2 \left(1 + \mathrm{SINR}_{ij}^{w}\right), \tag{4}$$

where w_i is the transmission bandwidth allocated by node i to the subtask to be offloaded.

Similarly, when the D2D mode is adopted and both sides during the communication process use an exclusive D2D frequency band (i.e. it can also be considered as the uplink frequency band of wireless mode with multiplexing), interference occurs due to the same reason. The communication rate in D2D mode is expressed as follows:

$$r_{i,j}^{d} = w_i \log_2 \left(1 + \mathrm{SINR}_{ij}^{d}\right), \tag{5}$$

where SINR_{ij}^{d} is the SINR between nodes i and j.

$$\mathrm{SINR}_{i,j}^{d} = \frac{P_i h_{i,j} s_{i,j}^{-\alpha}}{\sum\limits_{k \in \Xi \setminus i} P_k h_{k,j} s_{k,j}^{-\alpha} + N_1}, \tag{6}$$

and Ξ is the the node set using the same frequency band. The noise power in D2D communications mode is denoted as N_1.

The computing latency depends on the computational resources required by the subtask and the capability of the processor inside the edge node. This latency can be specifically expressed as $t_j^{c,m} = c_m / f_j$, where c_m is the required CPU cycles of subtask m and f_j is the the number of CPU cycles per second that the node j processor can run.

The waiting latency depends on the time the subtask spends in the waiting queue. When subtask m is placed at the end of the task waiting queue Q_j^w in node j, this time can be expressed as follows:

$$t_j^{w,m} = \sum_{n \in Q_j^w, n \neq m} t_j^{c,n}, \tag{7}$$

where $t_j^{c,n}$ is the computational latency of subtask n, which is the task ranked higher in Q_j^w than m.

In conclusion, the total latency t_i^m for subtask m of requester i is computed as follows:

$$t_i^m = max\{\zeta t_{ij}^{o,m}, max\{t_{kj}^{d,h}, k \in \mathbf{p}(m)\}\} + t_j^{w,m} + t_j^{c,m} \tag{8}$$

where i is the task generator node and j is the execution node (i.e. $d_j^m = 1$ and $\varsigma_j = 1$), k is the precursor of task m and $\zeta \in [0, 1]$ is the flag for identifying whether the task needs to receive offloading data from the requester i or not. Note that the delivery latency and offloading latency are 0 when $m = 0$ or $m = |\mathcal{T}|$.

The latency for the whole application is:

$$t_i^{\mathcal{G}} = \sum_{m=0}^{|\mathcal{T}|} t_i^m \tag{9}$$

Let the start time of task m be $t^{s,m}$; the end time is $t^{e,m} = t^{s,m} + t_i^m$. Analogously, the end time for the whole application is expressed as: $t^{e,\mathcal{G}} = t^{s,\mathcal{G}} + t_i^{\mathcal{G}}$.

2) Energy: Energy consumption is another overhead aspect of interest during the computing process. This includes the computing energy consumption, transmission energy, and energy consumed in the waiting queues.

First, the transmission energy consumption is related to the transmission power and transmission time. The energy consumption about subtask m are summarized as follows:

$$\begin{cases} e_{ij}^{o,m} = P_i t_{ij}^{o,m} \\ e_{kj}^{d,m} = P_k t_{kj}^{d,m} \\ e_{ji}^{r,m} = P_j t_{ji}^{r,m} \end{cases}, \tag{10}$$

where $e_{ij}^{o,m}$ is the energy consumed during offloading. $e_{kj}^{d,m}$ and $e_{ji}^{r,m}$ are the consumption during delivering and returning, respectively. For the convenience of expression, let $e_j^{t,m}$ represent the total transmission energy consumption during the process of executing subtask m, i.e.

$$e_j^{t,m} = \zeta e_{ij}^{o,m} + \sum_{k \in \mathbf{p}(m)} e_{kj}^{d,m} + e_{ji}^{r,m}. \tag{11}$$

The computational energy consumption is related to the attributes of the processor and required computing resources. Similar to reference [53], this energy is computed as:

$$e_j^{c,m} = \kappa_j q_m (f_j)^2 \tag{12}$$

where κ_j is a constant related to the device architecture and q_m is the input data size of the computation task m.

Last, but not the least important is the task idle energy consumption while waiting to be executed. As this value is almost negligible, we describe this energy as a constant: $e_j^{w,m}$.

Following summation, the energy consumed by task m requested by node i is:

$$e_i^m = e_j^{t,m} + e_j^{c,m} + e_j^{w,m}. \tag{13}$$

where i is the task generator and j is the execution node (i.e. $d_j^m = 1$ and $\varsigma_j = 1$). And the total application energy is:

$$e_i^{\mathcal{G}} = \sum_{m=0}^{|\mathcal{T}|} e_i^m. \tag{14}$$

In conclusion, the optimization problem of overhead for performing the computational dependent application $\mathcal{G}$ is:

$$\begin{aligned} minimize \quad & O_i^{\mathcal{G}} = t_i^{\mathcal{G}} + \varepsilon e_i^{\mathcal{G}} \\ s.t. \quad & d_j^m \in \{0,1\}, \sum_{j=0}^{|\mathcal{N}|} d_j^m = 1, \forall j \in \mathcal{N}, \forall m \in \mathcal{T} \\ & t_i^{s,m} > t_i^{e,k}, \forall k \in \mathbf{p}(m) \end{aligned} \tag{15}$$

where ε is the balance factor between latency and energy.

D. Problem Analysis

1) NP-hardness: The system model has been introduced. And there are several key obstacles to solve the optimization problem described in eq. (15). The complex dependency relations among subtasks greatly increase the difficulty. For example, the successor must be executed after all the predecessors and the delivery delays between the executive nodes of dependent subtasks need to be considered. The minimization problem (15) is NP-hard [22], so it cannot be solved optimally within a polynomial time. In this subsection, we analyze and prove the problem is NP-hard.

As discussed in [48], once a single problem is proved to be NP-hard, the procedure for proving additional problems are NP-hard is simple. For instance, given a problem ***A***, NP-hard proof involves showing that problem ***A*** can be transformed (polynomial) to an already known NP-hard problem ***B***. Thus, the NP-hardness proof process which will be employed in this paper consists of the following four steps: 1) Find problem ***B*** and show that ***B*** is NP-hard. 2) formulate the new problem ***A***. 3) construct a transformation f from ***A*** to ***B***. 4) prove that f is a polynomial transformation.

First, the definition of the well-known Traveling Salesman Problem is introduced.

Definition 1: Traveling Salesman Problem (TSP). Given a group of cities and distances between each pair of cities (i.e. an undirected complete graph), the traveling salesman needs to find the shortest path that traverses each city once and returns to the starting point. TSP is a well-known NP-hard problem [48].

Proof:

To prove the NP-hardness, we first consider a special case of our problem in eq. (15) (problem ***A***). In problem ***A***, we assume that each node in $\mathcal{N}$ can perform one subtasks at most during the offloading process. This special case is used to keep the components of eq. (15) consistent with the TSP problem (problem ***B***). In the TSP problem, the goal is to visit each city once without repetition. The difference between problems ***A*** and ***B*** is the number of subtasks $|\mathcal{T}|$ to process and required number of nodes N to traverse. Next, aim to perform polynomial problem transformation. There exist three cases for transformation from problem ***A*** to problem ***B***, namely:

- $N > |\mathcal{T}|$. The number of subtasks of ***A*** should be reduced to N.
- $N = |\mathcal{T}|$. The two problems are the same.
- $N < |\mathcal{T}|$. The number of subtasks of ***A*** should be increased to N.

Regardless of the case, it is obvious that the transformation is polynomial and, as according to [48] problem ***B*** is NP-hard, so is the special case problem ***A***. If a special case like ***A*** is NP-hard, the original problem formulated in eq. (15) is also NP-hard.

■

Additionally, interferences during communication and waiting queues within each node are updated continuously, resulting in significant dynamic characteristics of the environment. In this case, even if the same strategy is adopted for the same available nodes, the optimal offloading decision may be completely different. This suggests finding a different approach to solve this dynamic and complex coupling problem.

2) Subproblem De-composition: In eq. (15), the optimization objective mainly includes the overhead during data transmission and task execution. The objective is similar to the one in traditional task offloading, but the difference is in the dependency relation present in the constraint condition. This implies that the offloading order of subtasks needs to be decided, which also determines the environment available to every successor (i.e. remaining resources within nodes, network conditions, etc.). Therefore, the original problem from eq. (15) is actually broken down into two subproblems: subtask scheduling order (i.e. **dependent subtask scheduling**), and decision of the node to perform the offloaded subtask (i.e. **dependent subtask offloading**). These are addressed in sections IV and sections V, respectively.

III. Dependent Subtask Scheduling

Intuitively, the overhead is directly related to the offloading decision of each subtask. However, in fact, due to the existence of dependency relations, the subtask scheduling order is an important factor that affects the computational offloading and the order of delivery. Therefore, the scheduling problem of dependent subtasks is focused on in this section. Specifically, the subtask priority is addressed first and then the novel generator-executor Multi-Queue Priority (MQP) scheduling algorithm based on latest finished time is introduced to handle the

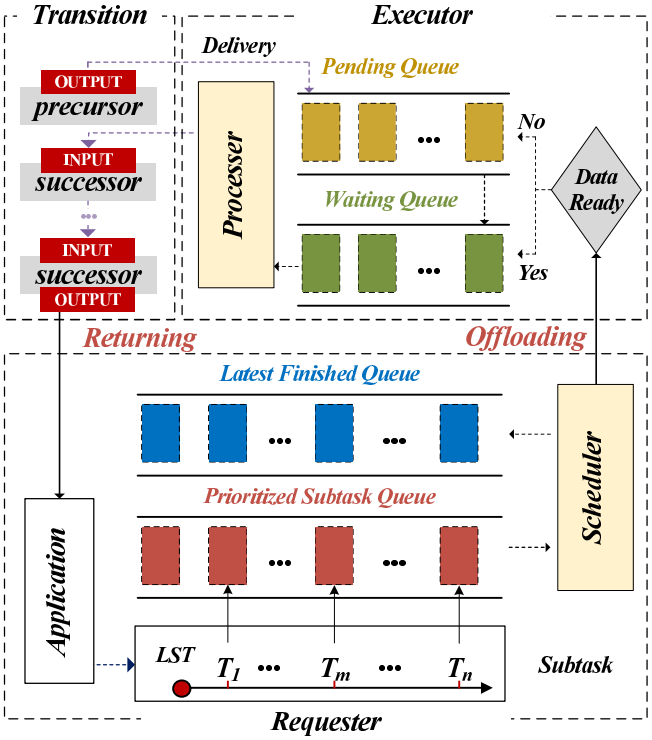


Fig. 3. Dependent Task Scheduling

offloading order of dependent subtask. Finally, the complexity is analyzed at the end of this section.

A. Associating Subtask Priority

The purpose of associating subtask priorities is to make the transition of subtask smoother and avoid the phenomenon related to a subsequent subtask having to wait for the delivery of output data from a precursor. This shortens the overall latency and helps complete the computational application within the deadline.

First, the latest start time (LST) of each subtask is defined as the lowest limit for scheduling. Any start time later than this value will lead to the overtime, i.e. the end time of the application $t^{e,\mathcal{G}}$ will exceed the threshold τ. LST is used to reflect the urgency of subtasks and judge their priorities. LST of task m is expressed as $t_i^{ls,m}$ in eq. (16), where t_i^m is defined in eq. (8).

$$t_i^{ls,m} = \begin{cases} \tau - t_i^m & m = |\mathcal{T}| \\ \min\limits_{n \in s(m)} \{t_i^{ls,n} - t_i^m\} & otherwise \end{cases} \tag{16}$$

Equation (16) shows the calculation process of the LST for a subtask. To be specific, for the final subtask, namely task $|\mathcal{T}|$, the LST is the latency limit of the application τ minus the latency associated with its execution (estimated by the generator device). It should be noted that the estimated latency is theoretical and mainly demonstrated the urgency of the subtask. Thus, $t_i^{ls,|\mathcal{T}|}$ is $\tau - t_i^m$. Noted that for any subtask m except $|\mathcal{T}|$, LST $t_i^{ls,m}$ is determined by the LST of its subsequent subtasks $n, \forall n \in \mathbf{s}(m)$. Therefore, as shown in equation (16), $t_i^{ls,m}$ is the LST of the successor subtask $t_i^{ls,n}$ minus the estimated time t_i^m of subtask m. Since any subsequent subtask can only be performed after the execution of the predecessor subtask m, subtask m must be started before the earliest LST of all subsequent subtasks. Function (i.e. $min\{\cdot\}$) is utilized in equation (16) to obtain the earliest LST. And the priority queue Q_i^p can be organized by sorting the subtasks in descending order of their LST.

B. Multi-Queue Priority Task Scheduling

The scheduling of tasks depends not only on the priority, but also on dependencies. One subtask cannot be selected until all of its precursor subtasks are completed. In other words, the computation of a subtask also requires all the data it expects to be delivered, before it can be executed. Therefore, in addition to maintaining the waiting queue Q^w, the execution node also needs to maintain another waiting delivery data queue Q^d(i.e. Pending Queue). The former is used to store the subtasks that have arrived, while the latter is used to maintain those subtasks from the former list which cannot be completed due to the lack of data. The process of scheduling these subtasks is detailed in Algorithm 1.

The MQP algorithm maintains three queues and an array. Q_i^p is used to store the subtasks in terms of their priority. Q_f stores the latest finished subtasks and Q_e records the current schedulable subtasks. In addition, array $\mathcal{F}$ contains the status of each subtask, with values of 0 for *waiting for execution* and 1 for *finished*. The MQP algorithm has four steps. First, the requester i generates the priority queue Q_i^p based on the application dependency graph. Second, the current executable task queue Q_e is generated based on the recently finished tasks Q_f. The next subtask to be scheduled is then selected according to subtask priorities. Finally, the data structure is updated. In this way, the update structure with space for time is implemented and the maintenance and computation latency of data can be reduced.

C. Complexity Analysis

This subsection analyses the time complexity of the MQP algorithm. First, the subtasks need to be traversed in the priority generation phase. This has a complexity of $O(Mlog_M), M = |\mathcal{T}|$ [54]. Then, the executable tasks generation depends on the number of latest finished subtasks and number of precursors to the subtask. As mentioned above, the latest finished subtasks refer to those subtasks which have been finished with not-yet-executed successors. In the breadth first search (BFS) graph algorithm, the successor node cannot be retrieved until the predecessor node is processed. As for the successor node with two or more predecessors, it will be retrieved and added to the queue when either of its predecessors pop from the front of the queue (this is more relaxed than the proposal). Therefore, the number of the nodes in the queue (i.e. unpopped) at any given time is the number of currently retrieved subtasks with unretrieved successor nodes, and is also the number of executable subtasks. Since it is related to graph BFS, we call the maximum length of the queue the maximum number of branches of the application graph and express it as K. On the other hand, the number of precursors and successors of a subtask can be regarded as the in-degree and out-degree of the node. An efficient graph structure can

Algorithm 1 Multi-Queue Priority Algorithm (MQP)

Initialization:
Q_i^p = [] Q_f = [] Q_e = [0] $\mathcal{F}$ = zeros(1, $|\mathcal{T}|$)
End Initialization
/* Task Priority Generate */
Input: *Dependency Task Graph* $\mathcal{G}$
for Sub-Task m in $\mathcal{G}$ **do**
  Compute the latest start time $t_i^{ls,m}$
  Q_i^p.*append*($t_i^{ls,m}$)
end for
while True **do**
  /* Executable Task Generation */
  Input: *Latest Finished Task Queue* Q_f, τ
  for $q_f \in Q_f$ **do**
    for $q_f^t \in s(q_f)$ **do**
      if $\mathcal{F}[q_f^t] == 0$ and $\mathcal{F}[q_f^{t,p}] == 1, \forall q_f^{t,p} \in p(q_f^t)$ **then**
        Q_e.*append*(q_f^t)
      end if
    end for
  end for
  /* Task Scheduling Selection */
  Input: *Waiting Execution Task Queue* Q_e, Q_p^i
  $T_p = -1, T_{p,max} = -1$
  for $q_e \in Q_e$ **do**
    if $Q_i^p[m] > T_{p,max}$ **then** $T_p = q_e$
      $T_{p,max} = Q_i^p[m]$
    end if
  end for
  /* Update Queue Q_f */
  Q_f.*append*(T_p)
  Q_e.*remove*(T_p)
  $\mathcal{F}[T_p] = 1$
  if $\mathcal{F}[T_p^{p,s}] = 1, \forall T_p^{p,s} \in s(T_p^p), \exists T_p^p \in p(T_p)$ **then**
    Q_f.*remove*(T_p^p)
  end if
end while

be represented with two orthogonal linked lists for each node precursors and successors. Therefore, the number of precursors and successors of any node can be obtained using $O(N)$ by getting the length of the appropriate linked list. If the length is stored, the complexity can be reduced to $O(1)$. The average number of precursors and successors is expressed as N.

The maximum number of branches and average number of precursors/successors of nodes are properties of the graph itself. In fact, these two values can be pre-fetched and included in the application information. The complexity of this step is $O(KN^2)$ [22]. The task selection is performed from the current executable subtask queue according to task priority. If the queue length is K, the task selection complexity is $O(K)$. Finally, the queue should be updated. The finished subtask is removed from the schedulable subtask queue. Then, the subtask is added to the latest finished subtask queue and the queue is updated next. For example, those subtasks, for which all their successors are finished, should be removed from the queue. Since only one subtask is scheduled at a time, the update process only needs to obtain the precursor list of that subtask and further determine whether all the successors have been finished. As the length of the precursor/successor list is N, the complexity of queue update process is $O(N^2)$. In conclusion, the overall complexity of the algorithm is $O(Mlog_M)+O(KN^2)+O(N)+O(K)+O(N^2) = O(M+KN^2)$.

IV. Model-free Dependent Task Offloading

As mentioned above, due to the coupling dependencies, the optimization problem eq. (15) is an NP-hard problem that is difficult to be solved within exponential time. Due to the highly dynamic fluctuations of the environment, the optimal solution found in the current time slot has a limited effect in time. Fortunately, model-free reinforcement learning emerges recently and provides an effective way. Different from model-based schemes, it does not need to model the environment, and obtains a large number of samples through continuous interaction with the external environment, thus approximating the strategy to the optimal solution. Benefit by the improvement of hardware performance, the training efficiency of model-free algorithm has been greatly improved. Next, the detailed solution is described in this section for solving the offloading decision problem and minimizing the long-term overhead. And the main idea is stated as follows.

First, the offloading process of dependent subtasks is considered as a Markov Decision Process (MDP). Then, the information required by decision-making is simplified from the graph structure to the vector structure of one subtask. The dependency relations between subtasks are further expressed as state transitions, which reflect the long-term benefit in the chain of execution. Finally, a deep reinforcement learning based on an Actor-Critic architecture is proposed to solve the minimization problem described in eq. (15).

A. MDP

MDP models the network environment as a time-discrete system and the requester is considered as an offloading agent. A MDP is commonly characterized as a 4-tuple $\mathbb{M} = \{\mathbb{S}, \mathbb{A}, \mathbb{P}, \mathbb{R}\}$, where $\mathbb{S}$ is the state space, $\mathbb{P}$ is the transition probability between different states describing the dynamic nature of the environment, $\mathbb{A}$ is the action space and $\mathbb{R}$ is the reward. As the state of the agent is high-dimensional and continuous and $\mathbb{P}$ cannot be accurately obtained, MDP is simplified as a model-free process $\mathbb{M} = \{\mathbb{S}, \mathbb{A}, \mathbb{R}\}$.

1) State Space $\mathbb{S}$*:* The state is represented in a vector form. It describes the status of the node, including task information and available computing resources. The task information includes all the subtasks and their dependencies (i.e. DAG $\mathcal{G}$) to facilitate decision-making. However, there are two obstacles of using these information directly. On one hand, larger vector dimension leads to an exponential expansion of the state space, which makes its traversal difficult. On the other hand, the decision-making of dependent subtasks of the application is a step by step process, and the offloading decision of the current step is directly related to its precursor, while the correlation

with other subtasks is weak. Therefore, we simplify the task information in terms of the current scheduled subtask. The state space can be denoted as $\mathbb{S} = \{\mathbb{T}, \mathbb{N}\}$, where $\mathbb{T}$ and $\mathbb{N}$ are the space of the schedulable subtasks and available nodes (i.e. candidate executor), respectively. Specifically, the state in slot t is denoted as $\mathcal{S}_t$.

2) Action Space $\mathbb{A}$*:* An action is the decision made by a requester with offloading strategy $\pi : \mathbb{A} \leftarrow \mathbb{S}$. It has a direct effect on the external environment. The action space is denoted as a $|\mathcal{N}|$ dimension vector and the action in slot t is denoted as $\mathcal{A}_t = \{d_1, ..., d_{|\mathcal{N}|}\}, d_i \in [0, 1], i \in \mathcal{N}$, which is a one-hot vector. Each element can have either a value of 0 or 1. The latter (i.e. $d_i = 1$) corresponds to the selection. There exists the following coupling relationship between elements: $\sum_{i=1}^{|\mathcal{N}|} d_i = 1$.

3) Reward $\mathbb{R}$*:* Intuitively, the reward $\mathbb{R} \leftarrow \mathbb{S} \times \mathbb{A}$ is the feedback obtained by the requester during offloading, which directly determines the strategy. The optimization goal in eq. (15) is to minimize the overall overhead of the application-level task, but in fact, the decisions at each step can only serve one subtask. Therefore, we represent the overhead of a single subtask m in slot t as $o_t^m = t_i^m + \varepsilon e_i^m$, with the terms from eq. (8) and eq. (13), respectively. As all subtasks are progressively completed, the total reward becomes the optimization objective in eq. (15).

The immediate reward of one step in slot t is denoted as $R_t(s, a) = -o_t^m - \mathbb{I}_{\{t,\tau\}}\Omega|_{\mathcal{S}_t=s,\mathcal{A}_t=a,s\in\mathbb{S},a\in\mathbb{A}}$, where $\mathbb{I}_{\{t,\tau\}}$ is the latency indicator if $t_i^m > \tau$ and Ω is the penalty term for timeout. However, due to the dynamic nature of the environment and dependency between subtasks, it is not sufficient to consider only the benefits of the current subtask to optimize the application-level overhead. The agent should evaluate the current action taken and judge it whether can provide more possible benefit behind the action to the future or not. Thus, the goal of the agent is changed to maximize the expected long-term cumulative return $\mathcal{R}_t(\pi)$, which is defined as follows.

$$\mathcal{R}_t(\pi) = \frac{1}{T-t} \lim_{T\to\infty} \sum_{t'=t}^{T} R_{t'}(s, a), \tag{17}$$

where π is the adopted offloading strategy. For the sake of clarity, the actual state, action and reward of agent are expressed as s, a, r, respectively, in the following text to distinguish it from state space, action space and reward and s_t, a_t, r_t are specified as the corresponding attributes of agent in slot t.

Based on the above definition, the action-value $Q_\pi \leftarrow \mathbb{S}\times\mathbb{A}$ is introduced in eq. (18).

$$Q_\pi(s, a) = \mathbb{E}[r_t + \sum_{t'=t}^{T} (\gamma^{t'-t} \cdot r_{t'}|s_{t'}, a_{t'})]|_{s_t=s,a_t=a} \tag{18}$$

where $s \in \mathbb{S}$ and $a \in \mathbb{A}$.$\gamma \in [0, 1]$ is a discount factor, indicating the impact of the current action into the future. The best action $a_t^* = \arg\max_{a\in\mathbb{A}} Q_\pi(s, a)$ can be selected with a greedy strategy.

B. Actor-Critic-based Subtask Offloading Algorithm

Actor-Critic [55] is a paradigm of deep reinforcement learning(DRL) [56]. On one hand, the strong fitting ability of neural network (NN) can adapt to the continuous state space, while the architecture has strong decision-making ability to give the optimal action. On other hand, it sets two roles of *Actor* and *Critic*, so that the balance between fast update and unbiased estimation can be achieved. Next, we introduce a two-phase subtask offloading algorithm based on *Actor-Critic*. These phases are offline training and online deployment, respectively.

Algorithm 2 Dependency Task Offloading

1: */* Offline Training Phase (with period p) */*
2: $t = 0$;
3: **while** $t \leq N$ **do**
4: ***Actor:***
φ_μ generates action a_t with exploration-exploitation strategy.
Agent executes action a_t and obtains the reward r_t.
The state of agent is transformed from s_t to s_{t+1}.
The transition tuple $\{s_t, a_t, r_t, s_{t+1}\}$ is restored.
φ'_μ generates the next action a_{t+1}.
5: ***Critic:***
θ_μ evaluates $Q_\pi(s_t, a_t)$ of the action a_t with current state s_t.
θ'_μ computes the target Q.
θ_μ computes the loss $L(\theta)$.
6: *The current networks in **Actor** and **Critic** (i.e., θ_μ and φ_μ) are updated.*
7: **if** $t\%T_u == 0$ **then**
*The target networks in **Actor** and **Critic** are updated as follows.*
$\theta'_\mu = \nu\theta_\mu + (1-\nu)\theta'_\mu$
$\varphi'_\mu = \nu\varphi_\mu + (1-\nu)\varphi'_\mu$
8: **end if**
9: **end while**

Fig. 4 illustrates the relationship between the training phase and the deployment phase. In the training phase, the agent interacts with the environment to obtain the state and generate the action. It can realize self-estimation through *Critic*, achieving rapid update. Meanwhile, one trained neural network(i.e. current network of *Actor*) participates in the deployment phase, which greatly reduces the size of the model to be issued.

1) Offline Training: Training module can run in edge servers or other computation-rich nodes. It maintains two components: *Actor* φ and *Critic* ϑ. Due to the deployment of duel network architecture, they include a total of four NNs (i.e. the current network φ_μ, ϑ_θ and the target network $\varphi_{\mu'}$ and $\vartheta_{\theta'}$). μ,μ',θ and θ' are the parameters of each NN, respectively. *Actor* is the policy network with input the state vector and is responsible for generating the action of offloading and interaction with the environment. *Critic* is the value network with input the concatenation from state and action and is responsible for evaluating the *Actor* decision. The choice for dual networks is in order to improve the stability of network convergence. The transitions that have been experienced are

stored in the replay buffer for training and updating. Finally, the networks are optimized through the back propagation of loss $L(\theta)$.

$$L^{(t)}(\theta) = E\left[\left(y(t) - Q_{\pi,t}(s,a;\theta)\right)\right]^2 |_{\mathcal{S}_t=s,\mathcal{A}_t=a}, \quad (19)$$

where $Q_{\pi,t}(s,a;\theta)$ is the Q value obtained from the current network of *Critic* and $y(t)$ is the target Q. Specifically, it is expressed as follows:

$$\begin{aligned} y(t) \quad &= E[(1-\beta)Q_{\pi,t}(s,a;\theta) + \beta R_t \\ &+\gamma Q_{\pi,t+1}(\mathcal{S}_{t+1},\mathcal{A}_{t+1};\theta')|\mathcal{S}_t = s, \mathcal{A}_t = a] \end{aligned}, \quad (20)$$

where $Q_{\pi,t+1}(\mathcal{S}_{t+1},\mathcal{A}_{t+1};\theta')$ is the evaluated Q value generated from the target network of *Critic*. β is the learning rate. It is worth noting that the parameters of the target network are smoothed periodically with the mode of soft update, i.e. $\theta'_\mu = \nu\theta_\mu + (1-\nu)\theta'_\mu$, $\varphi'_\mu = \nu\varphi_\mu + (1-\nu)\varphi'_\mu$, where ν is the smoothing coefficient. The smoothing coefficient makes agents learning process smoother and more stable, avoiding any temporary fluctuations due to sharp peaks in network environment or user requests.

The *Actor* is the policy network. Different from action evaluation, which can be quantified by the reward level, policy evaluation is more complex, because policy generates actions according to various states and obtains differentiated rewards, which is more abstract than action evaluation. The representation of performance objective J is given in the forms as follows.

$$J_\delta(\varphi_\mu) = \int_{\mathbb{S}} \rho^\delta(s)\, Q(s, \varphi_\mu(s)) ds = E_{s\sim\rho^\delta}\left[Q^\mu(s, \varphi(s))\right] \quad (21)$$

where ρ is used to represent the external environment, while δ represents the parameters of external environment attributes, which are only related to the environment and have nothing to do with the agent. Thus, ρ^δ is the instantiated external environment. $\rho^\delta(s)$ is denoted as the probability density function (PDF) of state and $Q(s, \varphi_\mu(s))$ is the Q value of the action generated from policy network μ. Commonly understood, J is the expected Q value brought by the policy.

The trained policy network φ_μ needs to meet the following objectives: $\mu = argmax_\mu J_\delta(\varphi_\mu)$ and the most effective training method is along the gradient descent direction of the policy function. In this respect, the Q value evaluated by the *Critic* is helpful and the policy gradient with respect to parameter μ can be expressed as:

$$\begin{aligned} \nabla_\mu J_\delta(\varphi) &\overset{(a)}{\approx} E_{s\sim\rho^\delta}\left[\nabla_a Q(s,a|\theta)|_{a=\varphi_\mu(s)} \cdot \nabla_\mu \varphi(s|\varphi^\mu)\right] \\ &\overset{(b)}{\approx} \frac{1}{\chi}\sum_{i=1}^{\chi}\left(\nabla_a Q(s,a|\theta)|_{s=\mathbf{s}_i, a=\varphi(\mathbf{s}_i)} \cdot \nabla_\mu \varphi(s|\varphi^\mu)|_{s=\mathbf{s}_i}\right) \end{aligned} \quad (22)$$

However, as mentioned earlier, ρ^δ in the environment is hard to be obtained. It is converted here to the expected value by the symbol of approximately equal (a). Further, it can be estimated by using the *Monte Carlo* method [58]. Specifically, the training model extracts a minibatch sample data from the replay buffer in each slot and obtains its average value as shown in the symbol of (b). The number of samples in the batch is denoted as χ. And soft update is performed every T_u step with the parameter of ν.

2) Online Deployment: Following the completion of the offline training process, the model can be delivered from the training platform Ψ to the edge nodes (e.g. laptop, phone, tablets, etc.) regularly during the off-peak periods of the request as shown in Fig. 4. The deployed agent only needs to include the current network module of the *Actor*, while having removed the exploration mechanism and maintained the greedy strategy to find the best offloading decision, i.e. $\pi = \varphi_\mu$. At the same time, the agent collects the application requirements of the requester. The dependency information is returned to the training platform when the devices acquire the updated model parameters, in order to improve the universality of the model. The details of the two-phased offloading process are shown in Algorithm 2. The running rounds of the algorithm are related to the number of subtasks within the application. Thus, its time complexity is $O(N)$.

V. Experimental Testing and Result Analysis

The experimental environment and relevant parameters are introduced first, followed by presentation of the comparison algorithm employed during testing. Experimental testing and their results are presented next in order to verify the feasibility and performance of the proposed solution.

A. Experimental Setup

1) Environment: We consider a test environment of 100*100 m^2, in which multiple edge nodes (i.e. laptops)

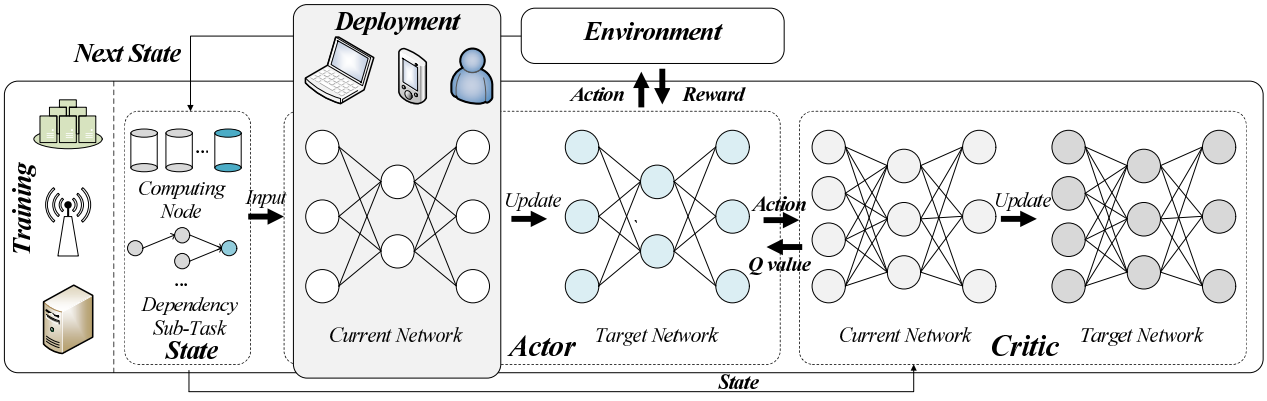


Fig. 4. Offloading Framework

TABLE II
EXPERIMENTAL PARAMETERS

Parameter	Value
Batch size	64
Layers of NN	3
Hidden layer dimension	64
Learning rate(Actor)	0.0001
Learning rate(Critic)	0.0001
Steps per update	60
Replay buffer length	1e6
Discount factor	0.99
The smoothing coefficient ν	0.005
Episode	800

are distributed according to the homogeneous Poisson point process (PPP) with spatial density ξ ($\xi = 8$ in this paper) as shown in Fig.5. The device nodes are marked with the serial number. Meanwhile, there is also a base station (BS) with edge server deployed in the scene, whose coordinates are randomly generated.

2) Parameter Configuration: The offloading program runs on a lab computer (Intel i7-7700k, Quad-Core 4.2Ghz/16GB). We assume that the computing resources are quantified by the number of CPU cycles under the same CPU architecture, which means a CPU cycle is equivalent to each device. The computing resources of devices are set to the [1,4] GHz interval, while those of BS is set to 10 GHz. In addition, the computing resource requirement of subtask obey a uniform distribution of [100,300] MHz. The input data size of each subtask is randomly selected in the [500,1500] KB interval and the output data size is subject to an uniform distribution between 50KB and 200KB. The deadlines of the different applications are randomly generated from [2,8] s. The number of subtasks is decided by the application and is randomly selected. On another hand, the transmit power of the nodes is set to 200 mW and the channel bandwidth is set to 20 MHz with 1200 subcarriers. The noise is 10^{-10} mW. Finally, the NN parameters involved during the training process are as indicated in Table II.

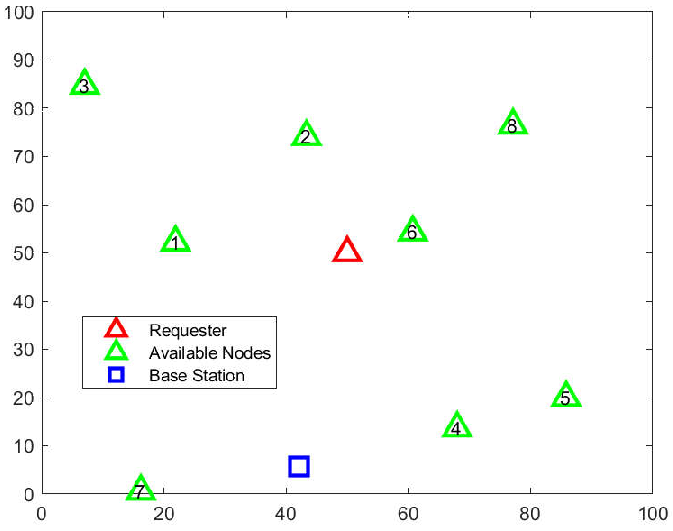


Fig. 5. Experimental environment

B. Comparison Algorithm

In order to compare the performance of our proposed scheme to those of other solutions, the following benchmark algorithms are selected:

1) Random Strategy: The requester randomly selects a schedulable subtask and offloads it on a feasible node.

2) Greedy Strategy: The requester preferentially schedules the tasks that require more resources and offloads them to the available nodes with the largest computing resources [50].

3) Local Computing: The dependent application computation tasks are computed locally without transmission and delivery [59].

4) Heuristic Algorithm: In literature [60], a genetic algorithm (GA) is utilized to solve the problem. The offloading decision is represented as a gene, and the action space is explored through operations, e.g. crossover, mutation, etc. When the average fitness of the population did not change after N rounds (i.e. we set N = 15 in our tests) or when the total number of iterations exceeds M rounds (i.e. we set M = 50 in our tests), the evolution stops and the best gene found (i.e. with the highest fitness) is regarded as the final decision.

C. Performance Evaluation

For performance evaluation of the proposed scheme, we carry out specific tests, mainly to assess the convergence of the algorithm, latency and energy consumption.

We first verify the convergence of the scheme in the training process and the performance of the deployment process. As shown in Fig. 6, the training process converges at $t = 100$, and a relatively stable jitter is maintained from that moment on. This indicates that the agent has developed a mature strategy and has implemented it into its offloading decision. During painting, 10 adjacent data points on the time dimension are grouped and the average value within the group is calculated as the new data point. The upper limit of the shaded part represents the maximum value in the group, and the lower limit represents the minimum value. This more clearly shows the fluctuation of data. In the training phase, the fluctuation is caused by the exploration strategy and dynamic environment. In the deployment phase, the exploration strategy and *Critic* are removed. Thus, the solution can always adopt strategic

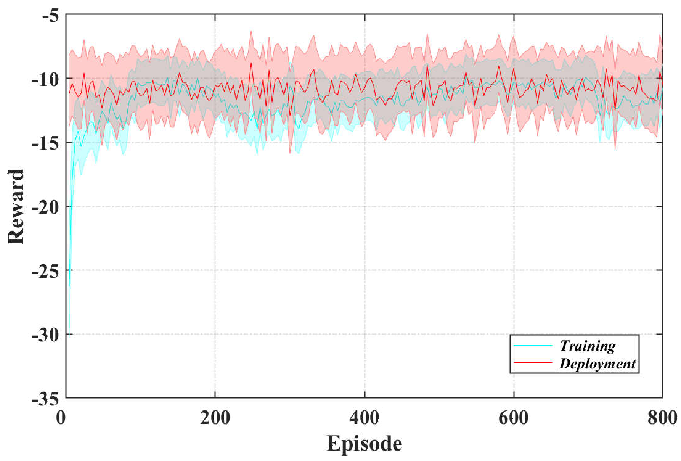


Fig. 6. Convergence

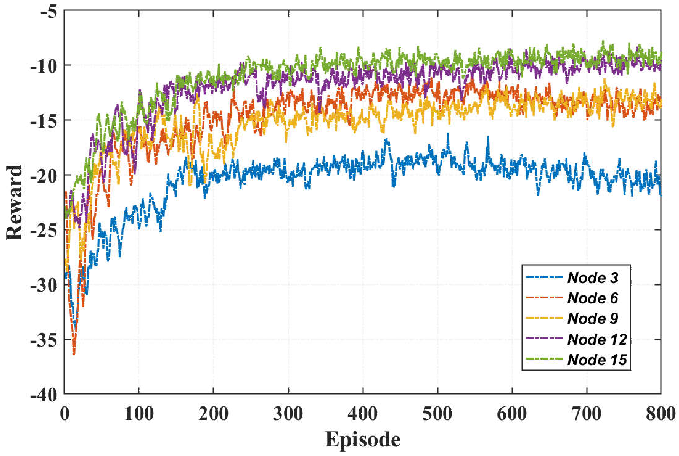


Fig. 7. Reward versus Node Number

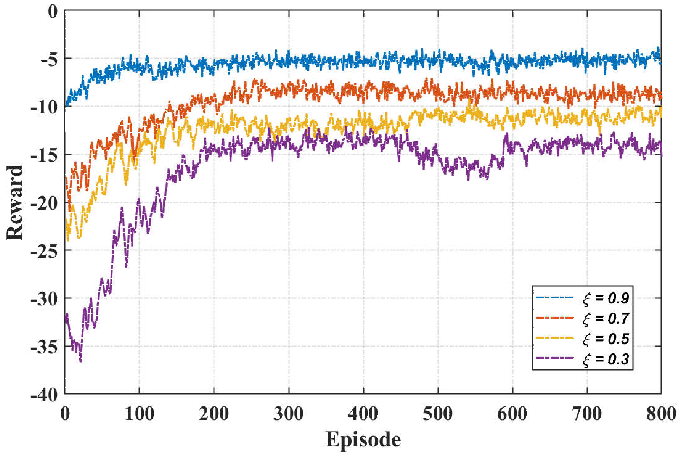


Fig. 8. Reward versus Node Density

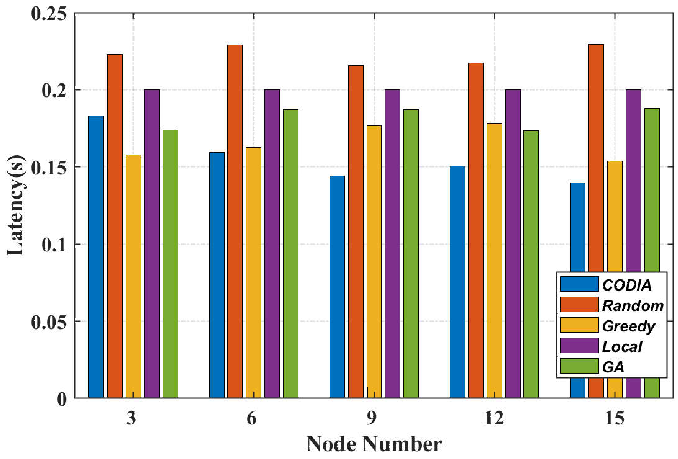


Fig. 9. Latency Performance

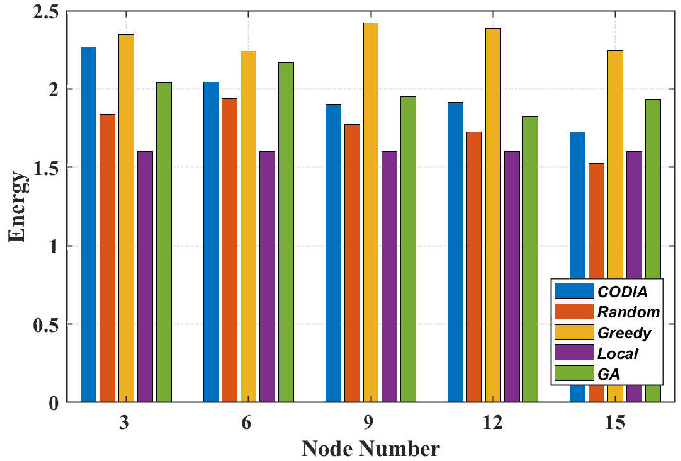


Fig. 10. Energy Performance

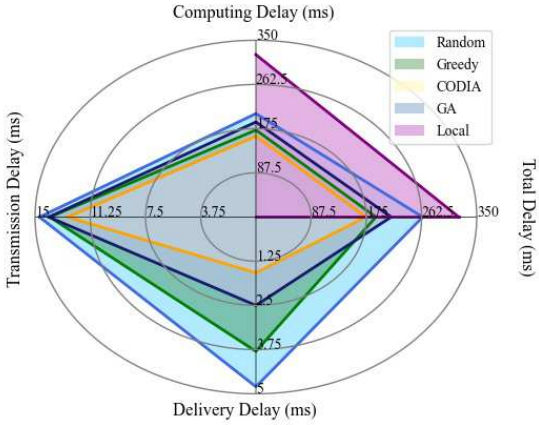


Fig. 11. Detailed Latency

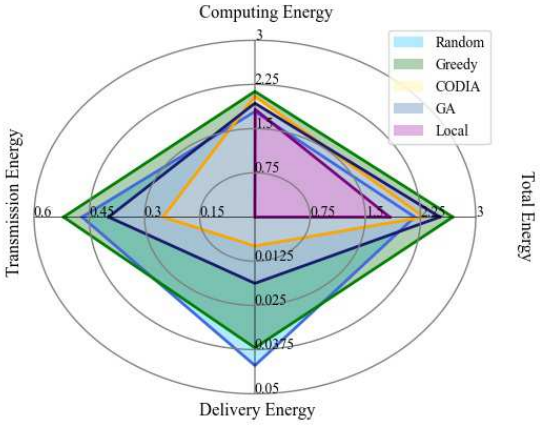


Fig. 12. Detailed Energy

optimal actions and achieve a little higher reward with more fluctuations.

Then, we adjust the number and the density of nodes, respectively, and observe the effects on the algorithm. During this process, a total of 800 episodes were run, and a graph structure application is completed in each episode. In Fig. 7, the reward always converges in the end and the return is different. In general, the benefits in situations with fewer nodes are less than those with more nodes. For example, the reward with 3 nodes is 50% less than the return with 15 nodes. At the

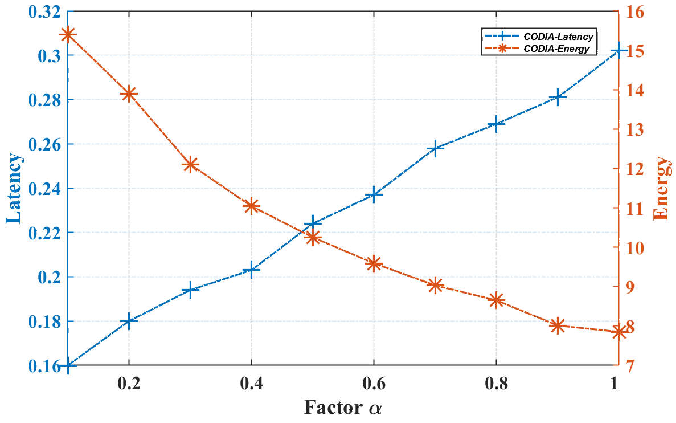


Fig. 13. Balance Factor α

same time, the effect of the number of nodes on the results tends to disappear eventually. The reward with 12 nodes and 15 nodes are basically identical. This is because the number of nodes has been able to support the maximum parallelism of the algorithm, and even if the number continues to increase, there will be an upper limit for the reward. Meanwhile, as shown in Fig. 8, the density has a similar effect on the payoff of the algorithm. With the increase of node density, the performance tends to improve. It can also be noted that the starting point of training varies with density. The reason can be attributed to the fact that the density directly affects the delivery and transmission process. The higher density avoids long-distance transmission and thus improves the final revenue.

On the other hand, in order to observe the performance of our proposed solution, we compare it with three advanced scheduling schemes and record the performance differences. In this test, we adjust the test parameters and observe how much revenue the requester can get. As shown in Figure 9, on one hand, when the number of nodes is small, the delay of our scheme is slightly higher than those of the greedy strategy and GA, but better than those of other schemes. At the same time, the latency decreases as the number of schedulable nodes increases, due to the benefits of utilizing the parallelism of subtasks. The results show that the delay of this method is 25% less than that of local computing. For GA, it can achieve superior performance when the number of nodes is small, but its performance decreases with the increase in the number of nodes. This is because GA cannot search enough in the action space to find the optimal solution in limited time. Therefore, when the number of nodes increases, the algorithm often converges to a local optimal solution. When the number of nodes reaches 15, the performance of GA is significantly lower than that experienced by both the proposed scheme and greedy strategy.

Fig. 10 illustrates the energy consumption across the solution tested. When the number of nodes is small, the transmission and delivery occurred in frequent offloading makes *CODIA* energy consumption slightly higher than those of other schemes. As the number of nodes increases, the average energy consumption decreases due to CODIA's mechanism of saving the energy during the delivery process. The greedy strategy always offloads tasks to the node with the largest computing resources. Although the delivery energy consumption is reduced, the increase in computation energy consumption makes the algorithm always consume the most energy. For local computing, as there is no content transmission and delivery process, and computing power is limited, less energy is consumed. Finally, the GA performance of controlling energy consumption is poor. When the number of node is small, frequent deliveries increased the energy consumption. When the number of nodes is large, GA finds difficult to obtain the optimal solution in time. According to our test, it is expected to achieve best performance with 7 nodes under the finite time limit. Random strategies increase the energy consumption associated with the delivery process because the scheduling of subtasks is scattered in irregular locations.

In order to identify the source of the overhead in terms of latency and energy, we analyze each step of the offloading process in details. As shown in Fig. 11 and Fig. 12, latency and energy consumption are divided into three stages: computation, transmission and delivery. For local computing, as there are neither transmission nor delivery, the energy consumption only refers to computation energy, which is relatively small. In contrast, the time delay is relatively high. The random strategy can reduce the time delay to a certain extent, but

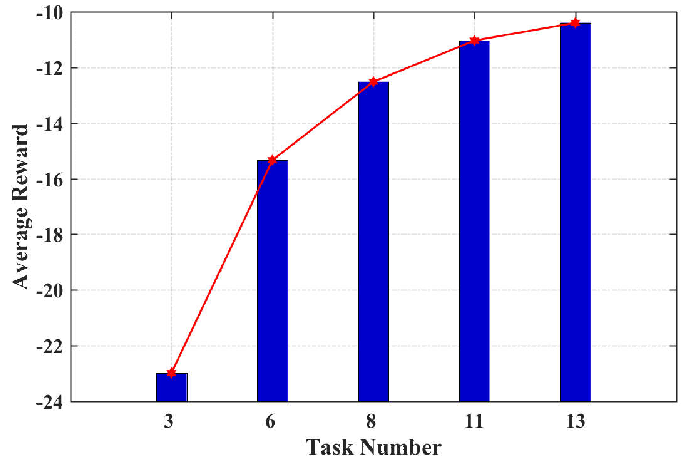


Fig. 14. Reward versus Task Number

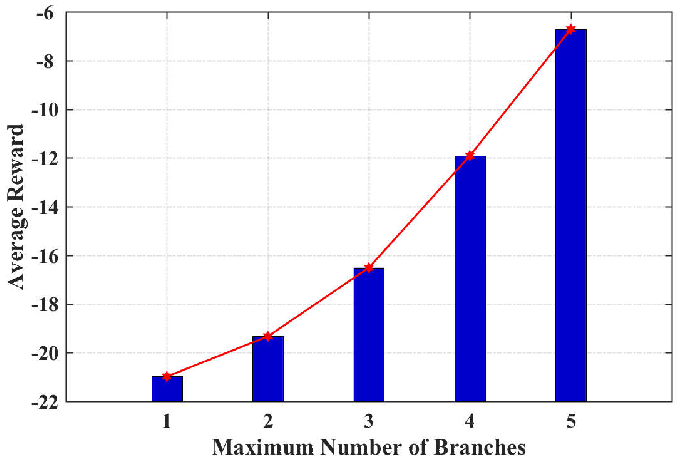


Fig. 15. Reward versus Branches Number

it has no outstanding characteristics in terms of both energy consumption and time delay. The time delay when the greedy strategy is employed is very close to that of the proposed solution. Unfortunately the greedy strategy also has the highest energy consumption. Regarding *GA*, there is a certain gap between *CODIA* and *GA* in the considered aspects, which can be understood as the latter fails to converge to the optimal solution in finite time. Therefore, the performance of delay and energy consumption are both poorer in comparison with those of the proposed scheme. Finally, the tests performed have shown that the proposed solution has better results in terms of both the delivery delay and energy consumption in comparison with the alternative solutions considered.

Next, we adjust some application-oriented parameters and observe their effect on the algorithm. In Fig. 13, the balance factor α is set in the interval of $[0.1, 1]$ and the latency is increasing with the decrease of energy consumed. This indicates that the proposed scheme can flexibly adjust its strategy according to the demands of time-sensitive applications and energy-saving applications.

In addition, when we held the other parameters constant and adjusted the number of tasks within the application, we observed the average gain of individual subtasks during offloading. As shown in Fig. 14, when the number of subtasks is 3, the reward is significantly lower than that for other applications, because the number of subtasks is small and the execution process is linear. At the same time, with the same offloading strategy, the reward of the task will increase with the number of the subtasks, which indicates that it can potentially affect the parallelism of the tasks.

Meanwhile, we adjusted the branches number inside the application. As Fig. 15 shows, the average reward increases significantly as the number of branches increases. In sharp contrast to Fig. 14, the exponential growth of the number of branches brings a higher growth rate than the logarithmic growth of the number of tasks. This indicates that compared with the former, the latter is the direct factor, affecting the parallelism of computation.

VI. Conclusions

In this paper, we propose an innovative solution with edge intelligence for computational tasks offloading for dependent IoT applications (i.e. CODIA). First, we propose a priority-based dependent subtask scheduling strategy, and analyze its algorithm complexity. On this basis, the computing offloading algorithm based on deep reinforcement learning is designed, which realizes the efficient offloading of computing tasks through offline training and online deployment. Finally, after a series of experimental tests, the results show that the proposed scheme has superior convergence and can reduce the time delay by 25% compared with the local computing while controlling the energy consumption. In the future, we will consider the next step from two aspects. On one hand, the more complex multi-agent environment will be included, which will bring exponential growth to the state space, making the training and deployment of agents extremely challenging. On the other hand, federated learning can protect the privacy of users during the process of training, which is also a field worth studying in the future.

References

[1] L. Yang, H. Zhang, M. Li, J. Guo and H. Ji, "Mobile Edge Computing Empowered Energy Efficient Task Offloading in 5G," *IEEE Trans. on Veh. Technol.*, vol. 67, no. 7, pp. 6398-6409, July 2018.

[2] APP Annie, Mobile Market Data Report 2021, App Annie, San Francisco, USA, 2021. Available:https://www.appannie.com/cn/insights/market-data/mobile-2021-new-records-beckon/

[3] L. Xiao, X. Lu, T. Xu, X. Wan, W. Ji and Y. Zhang, "Reinforcement Learning-Based Mobile Offloading for Edge Computing Against Jamming and Interference," *IEEE Trans. Commun.*, vol. 68, no. 10, pp. 6114-6126, Oct. 2020.

[4] X. Li,Y. Qin, H.Zhou,Z. Zhang,An intelligent collaborative inference approach of service partitioning and task offloading for deep learning based service in mobile edge computing networks,*Trans. Emerging Tel. Tech.*,2021;32:e4263.

[5] A. Lakhan,MA. Mohammed,S. Kozlov, JJPC. Rodrigues.Mobile-fog-cloud assisted deep reinforcement learning and blockchain-enable IoMT system for healthcare workflows, *Trans. Emerging Tel. Tech.*,2021;e4363.

[6] F. Xu,Z. Zhang,J. Feng,Z. Qin,Y. Xie,Efficient deployment of multi-UAV assisted mobile edge computing: A cost and energy perspective,*Trans. Emerging Tel. Tech.*,2022; e4453.

[7] S. Yang, J. Hu, K. Jiang, H. Xiao, M. Wang, Hybrid-360: An adaptive bitrate algorithm for tile-based 360 video streaming, *Trans. Emerging Tel. Tech.*, 2021;e4430.

[8] M. Golkarifard, J. Yang, Z. Huang, A. Movaghar and P. Hui, "Dandelion: A Unified Code Offloading System for Wearable Computing," *IEEE. Trans. Mob. Comput.*, vol. 18, no. 3, pp. 546-559, 1 March 2019.

[9] M. Mukherjee et al., "Latency-Driven Parallel Task Data Offloading in Fog Computing Networks for Industrial Applications," *IEEE Trans. Ind. Inform.*, vol. 16, no. 9, pp. 6050-6058, Sept. 2020.

[10] Y. Wang et al., "A Game-Based Computation Offloading Method in Vehicular Multiaccess Edge Computing Networks," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4987-4996, June 2020.

[11] K. Yang, S. Ou, H. Chen, "On effective offloading services for resource-constrained mobile devices running heavier mobile Internet applications," *IEEE Commun. Mag.*, vol. 46, no.1, pp. 56-63, January 2008.

[12] K. Wang, K. Yang and C. S. Magurawalage, "Joint Energy Minimization and Resource Allocation in C-RAN with Mobile Cloud," *IEEE Trans. Cloud Comput.*, vol. 6, no. 3, pp. 760-770, 1 July-Sept. 2018.

[13] M. Qin; N. Cheng; Z. Jing; T. Yang; W. Xu; Q. Yang; R. R. Rao, "Service-Oriented Energy-Latency Tradeoff for IoT Task Partial Offloading in MEC-Enhanced Multi-RAT Networks," *IEEE Internet Things J.*, vol. 8, no. 3,pp. 1896-1907, Feb. 2018.

[14] A. A. Ashraf Ateya, A. Muthanna, R. Kirichek, M. Hammoudeh and A. Koucheryavy, "Energy- and Latency-Aware Hybrid Offloading Algorithm for UAVs," *IEEE Access*, vol. 7, pp. 37587-37600, 2019.

[15] R. Yadav, W. Zhang, O. Kaiwartya, H. Song and S. Yu, "Energy-Latency Tradeoff for Dynamic Computation Offloading in Vehicular Fog Computing," *IEEE Trans. on Veh. Technol.*, vol. 69, no. 12, pp. 14198-14211, Dec. 2020.

[16] K. Wang, K. Yang, H. Chen and L. Zhang, "Computation Diversity in Emerging Networking Paradigms," *IEEE Wirel. Commun.*, vol. 24, no. 1, pp. 88-94, February 2017.

[17] J. Yan, S. Bi, L. Huang and Y. A. Zhang, "Deep Reinforcement Learning Based Offloading for Mobile Edge Computing with General Task Graph," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Dublin, Ireland, 2020, pp. 1-7.

[18] J. Yan, S. Bi and Y. J. A. Zhang, "Offloading and Resource Allocation With General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 8, pp. 5404-5419, Aug. 2020, doi: 10.1109/TWC.2020.2993071.

[19] Y. Han, Z. Zhao, J. Mo, C. Shu and G. Min, "Efficient Task Offloading with Dependency Guarantees in Ultra-Dense Edge Networks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1-6.

[20] S. Pan, Z. Zhang, Z. Zhang and D. Zeng, "Dependency-Aware Computation Offloading in Mobile Edge Computing: A Reinforcement Learning Approach," *IEEE Access*, vol. 7, pp. 134742-134753, 2019.

[21] J. Yan, S. Bi and Y. A. Zhang, "Optimal Offloading and Resource Allocation in Mobile-Edge Computing with Inter-User Task Dependency," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Abu Dhabi, United Arab Emirates, 2018, pp. 1-8.

[22] Y. Liu et al., "Dependency-Aware Task Scheduling in Vehicular Edge Computing," *IEEE Internet Things J.*, vol. 7, no. 6, pp. 4961-4971, June 2020.

[23] G. Zhao, H. Xu, Y. Zhao, C. Qiao and L. Huang, "Offloading Tasks With Dependency and Service Caching in Mobile Edge Computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 11, pp. 2777-2792, 1 Nov. 2021.

[24] M. Mehrabi, S. Shen, V. Latzko, Y. Wang and F. H. P. Fitzek, "Energy-Aware Cooperative Offloading Framework for Inter-dependent and Delay-sensitive Tasks," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Taipei, Taiwan, 2020, pp. 1-6.

[25] Y. Kao, B. Krishnamachari, M. Ra and F. Bai, "Hermes: Latency Optimal Task Assignment for Resource-constrained Mobile Computing," *IEEE. Trans. Mob. Comput.*, vol. 16, no. 11, pp. 3056-3069, 1 Nov. 2017, doi: 10.1109/TMC.2017.2679712.

[26] N. Eshraghi and B. Liang, Joint offloading decision and resource allocation with uncertain task computing requirement, in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Paris, France, 2019, pp. 14141422.

[27] Z. Meng, H. Xu, L. Huang, P. Xi, and S. Yang, Achieving energy efficiency through dynamic computing offloading in mobile edgeclouds, in *Proc. IEEE 15th Int. Conf. Mobile Ad Hoc Sensor Syst.*, Chengdu, China, 2018, pp. 175-183.

[28] A. Feriani and E. Hossain, "Single and Multi-Agent Deep Reinforcement Learning for AI-Enabled Wireless Networks: A Tutorial," *IEEE Commun. Surveys Tuts.*, March, 2021.

[29] T. Qiu, J. Chi, X. Zhou, Z. Ning, M. Atiquzzaman and D. O. Wu, "Edge Computing in Industrial Internet of Things: Architecture, Advances and Challenges," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2462-2488, Fourthquarter 2020.

[30] Y. Shi, K. Yang, T. Jiang, J. Zhang and K. B. Letaief, "Communication-Efficient Edge AI: Algorithms and Systems," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2167-2191, Fourthquarter 2020.

[31] H. Flores et al., Evidence-Aware Mobile Computational Offloading, *IEEE. Trans. Mob. Comput.*, vol. 17, no. 8, , 1 Aug. 2018.

[32] M. H. Chen, M. Dong, and B. Liang, Resource sharing of a computing access point for multi-user mobile cloud offloading with delay constraints, *IEEE. Trans. Mob. Comput.*, vol. 17, no. 12, Dec. 2018.

[33] F. Song, H. Xing, S. Luo, D. Zhan, P. Dai and R. Qu, "A Multiobjective Computation Offloading Algorithm for Mobile-Edge Computing," *IEEE Internet Things J.*, vol. 7, no. 9, Sept. 2020, pp. 8780-8799,

[34] H. Xiao, C. Xu, T. Cao, L. Zhong and G. Muntean, "GTTC: A Low-Expenditure IoT Multi-Task Coordinated Distributed Computing Framework with Fog Computing," in *Proc. IEEE Global Telecommun. Conf. (GLOBECOM)*, Waikoloa, HI, USA, 2019, pp. 1-6.

[35] T. Cao, C. Xu, J. Du, Y. Li, H. Xiao,C. Gong, L. Zhong, D. Niyato, "Reliable and Efficient Multimedia Service Optimization for Edge Computing-Based 5G Networks: Game Theoretic Approaches," *IEEE Trans. Netw. Serv. Manag.*, vol. 17, no. 3, pp. 1610-1625, Sept. 2020.

[36] X. Chen, Changqiao Xu, M. Wang, Z. Wu, L. Zhong and L. A. Grieco, "Augmented Queue-based Transmission and Transcoding Optimization for Livecast Services Based on Cloud-Edge-Crowd Integration," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 30, no. 11, pp. 4470-4484, Nov. 2021.

[37] M. Liu and Y. Liu, "Price-Based Distributed Offloading for Mobile-Edge Computing With Computation Capacity Constraints," *IEEE Wirel. Commun. Lett.*, vol. 7, no. 3, pp. 420-423, June 2018.

[38] C. You, K. Huang, H. Chae and B. Kim, "Energy-Efficient Resource Allocation for Mobile-Edge Computation Offloading," *IEEE Trans. Wirel. Commun.*, vol. 16, no. 3, pp. 1397-1411, March 2017.

[39] X. Qiu, L. Liu, W. Chen, Z. Hong and Z. Zheng, "Online Deep Reinforcement Learning for Computation Offloading in Blockchain-Empowered Mobile Edge Computing," *IEEE Trans. on Veh. Technol.*, vol. 68, no. 8, pp. 8050-8062, Aug. 2019.

[40] Q. Qi et al., "Knowledge-Driven Service Offloading Decision for Vehicular Edge Computing: A Deep Reinforcement Learning Approach," *IEEE Trans. on Veh. Technol.*, vol. 68, no. 5, pp. 4192-4203, May 2019.

[41] M. Tang and V. W. S. Wong, "Deep Reinforcement Learning for Task Offloading in Mobile Edge Computing Systems," *IEEE. Trans. Mob. Comput.*, Early Access, Nov. 2020.

[42] H. Zhou, X. Chen, S. He, J. Chen and J. Wu, "DRAIM: A Novel Delay-Constraint and Reverse Auction-Based Incentive Mechanism for WiFi Offloading," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 4, pp. 711-722, April 2020.

[43] Technical Specification Group Core Network and Terminals; Proximity-services (ProSe) User Equipment (UE) to Proximity-services (ProSe) Function Protocol Aspects; Stage 3 (Release 12), *3GPP TS 24.334 V1.1.0*, Jul. 2014.

[44] A. Orhean, F. Pop, I. Raicu, "New scheduling approach using reinforcement learning for heterogeneous distributed systems," *J. Parallel Distrib. Comput.*, vol. 117, pp. 292-302, 2018

[45] Z. Tang, J. Lou, F. Zhang and W. Jia, "Dependent Task Offloading for Multiple Jobs in Edge Computing," in *Proc. of 29th Int. Conf. Comput. Commun. and Networks (ICCCN)*, Honolulu, HI, USA, 2020, pp. 1-9.

[46] Asghari, A., Sohrabi, M.K. and Yaghmaee, F. Online scheduling of dependent tasks of clouds workflows to enhance resource utilization and reduce the makespan using multiple reinforcement learning-based agents. *Soft Comput.*, 24, 1617716199 (2020).

[47] H. Hao, Changqiao Xu, L. Zhong and G.-M. Muntean, "A Multi-update Deep Reinforcement Learning Algorithm for Edge Computing Service Offloading," in *Proc. of the 28th ACM Int. Conf. on Multimedia (ACM Multimedia)*, Seattle, United States, October 2020.

[48] Garey M R, Johnson D., Computer and Intractability: A Guide to the Theory of NP-Completeness. *New York: Freeman*, 1979.

[49] Y. Yao, J. Wang, B. Sheng, J. Lin and N. Mi, "HaSTE: Hadoop YARN Scheduling Based on Task-Dependency and Resource-Demand," in *Proc. IEEE 7th Int. Conf. Cloud Comput.*, Anchorage, AK, USA, 2014, pp. 184-191.

[50] S. Sundar and B. Liang, "Offloading Dependent Tasks with Communication Delay and Deadline Constraint," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Honolulu, 2018, pp. 37-45.

[51] Q. Li, M. Wen, S. Dang, E. Basar, H. V. Poor and F. Chen, "Opportunistic Spectrum Sharing Based on OFDM With Index Modulation," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 1, pp. 192-204, Jan. 2020.

[52] K. Zhang, Y. Zhu, S. Leng, Y. He, S. Maharjan, and Y. Zhang, Deep learning empowered task offloading for mobile edge computing in urban informatics, *IEEE Internet Things J.*, vol. 6, no. 5, pp. 76357647, Oct. 2019.

[53] Y. Mao, C. You, J. Zhang, K. Huang and K. B. Letaief, "A Survey on Mobile Edge Computing: The Communication Perspective," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 4, pp. 2322-2358, Fourthquarter 2017.

[54] R. Bleuse, S. Hunold, S. Kedad-Sidhoum, F. Monna, G. Mouni and D. Trystram, "Scheduling Independent Moldable Tasks on Multi-Cores with GPUs," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 9, pp. 2689-2702, 1 Sept. 2017.

[55] J. Luo, F. R. Yu, Q. Chen and L. Tang, "Adaptive Video Streaming With Edge Caching and Video Transcoding Over Software-Defined Mobile Networks: A Deep Reinforcement Learning Approach," *IEEE Trans. Wirel. Commun.*, vol. 19, no. 3, pp. 1577-1592, March 2020.

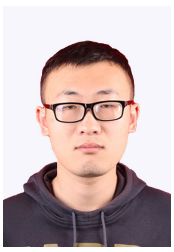
[56] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves ,I. Antonoglou, D. Wierstra and M. Riedmiller,"Playing Atari with Deep Reinforcement Learning", in *Proc. Neural Info. Proc. Syst. (NeurIPS)*, 2013.

[57] T. Degris, M. White and R. S. Sutton, "Off-Policy Actor-Critic", arXiv:1205.4839, *arXiv preprint*, 2013.

[58] Timothy P. Lillicrap, Jonathan J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, D. Wierstra, "Continuous control with deep reinforcement learning", in *Proc. Int. Conf. on Learning Repres. (ICLR)*, 2016.

[59] D. Dhiyagu and R. Shanmughasundaram, "Dependency and utilization aware Task Allocation for Multi-core Embedded Processors," in *Proc. Inno. in Power and Advanced Comput. Technolo. (i-PACT)*, Vellore, India, 2019, pp. 1-5.

[60] A. A. Al-Habob, O. A. Dobre, A. G. Armada and S. Muhaidat, "Task Scheduling for Mobile Edge Computing Using Genetic Algorithm and Conflict Graphs," *IEEE Trans. on Veh. Technol.*, vol. 69, no. 8, pp. 8805-8819, Aug. 2020.



Han Xiao received the B.E. degree in computer science from the Jinan University, in 2017. He is currently pursuing the Ph. D degree in the Network Architecture Research Center, Beijing University of Posts and Telecommunications (BUPT), advised by Prof. Changqiao Xu. His research interests include reinforcement learning, multimedia communications and panoramic video transmission.

Changqiao Xu (Senior Member, IEEE) received the Ph.D. degree from the Institute of Software, Chinese Academy of Sciences (ISCAS) in Jan. 2009. He was an Assistant Research Fellow and R&D Project Manager in ISCAS from 2002 to 2007. He was a researcher at Athlone Institute of Technology and Joint Training PhD at Dublin City University, Ireland during 2007-2009. He joined Beijing University of Posts and Telecommunications (BUPT), Beijing, China, in Dec. 2009. Currently, he is a Professor with the State Key Laboratory of Networking and Switching Technology, and Director of the Network Architecture Research Center at BUPT. His research interests include Future Internet Technology, Mobile Networking, Multimedia Communications, and Network Security. He has edited two books and published over 200 technical papers in prestigious international journals and conferences, including IEEE/ACM TON, IEEE TMC, IEEE INFOCOM, ACM Multimedia etc. He has served a number of international conferences and workshops as a Co-Chair and TPC member. He is currently serving as the Editor-in-Chief of Transactions on Emerging Telecommunications Technologies (Wiley). He is Senior member of IEEE.

Yunxiao Ma received the B.E. degree in telecommunications engineering from the School of Electronic Information Engineering, Inner Mongolia University, in 2019. She is currently pursuing the Ph.D. degree with the Network Architecture Research Center, School of Computing, Beijing University of Posts and Telecommunications, under the supervision of Prof. Changqiao Xu. Her research interests include multimedia communications, 360 degree video transmission and stochastic optimization.

Shujie Yang received the Ph.D. degree from the Institute of Network Technology, Beijing University of Posts and Telecommunications, Beijing, China, in 2017. He is currently a lecturer with the State Key Laboratory of Networking and Switching Technology. His major research interests include wireless communications and multimedia communications.

Lujie Zhong received the Ph.D. degree from the Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China, in 2013. She is currently an Associate Professor with the Information Engineering College, Capital Normal University, Beijing, China. She has published papers in prestigious international journals and conferences in the related area, including IEEE Communication Magazine, IEEE Transactions on Mobile Computing, IEEE Transactions on Multimedia, IEEE Internet Things Journal, IEEE INFOCOM and ACM Multimedia, etc. Her research interests include communication networks, computer system and architecture, and mobile Internet technology

Gabriel-Miro Muntean (Senior Member, IEEE) is a Professor with the School of Electronic Engineering, Dublin City University (DCU), Ireland, and co-Director of DCU Performance Engineering Laboratory. He has published 4 books and over 450 papers in top international journals and conferences. His research interests include rich media delivery quality, performance, and energy-related issues, technology enhanced learning, and other data communications in heterogeneous networks. He is an Associate Editor of the IEEE Transactions on Broadcasting, the Multimedia Communications Area Editor of the IEEE Communications Surveys and Tutorials, and reviewer for important international journals, conferences, and funding agencies. He coordinated the EU project NEWTON and leads the DCU team in the EU project TRACTION.