Journal of Computing and Electronic Information Management
ISSN: 2413-1660 | Vol. 13, No. 1, 2024

Pruning-based Deep Reinforcement Learning for Task Offloading in End-Edge-Cloud Collaborative Mobile Edge Computing

Hao Yang [a], Huifu Zhang *, Fan Luo [b], Fangjun Liu [c], Hao Chen [d]

School of Computer Science and Engineering, Hunan University of Science and Technology, Xiang tan, China

[*] **Corresponding author**: Huifu Zhang (Email: hfzhang@hnust.edu.cn); [a] yanghaohnust@163.com, [b] 22010501023@mail.hnust.edu.cn, [c]1989036826@qq.com; [d] 2184213023@qq.com

Abstract: The task offloading of Mobile Edge Computing (MEC) brings infinite possibilities to compute-intensive and latency-sensitive mobile applications. However, the dynamic nature of MEC systems and complex dependencies between computational tasks pose significant challenges to offloading decisions. In this paper, we address the task offloading problem of end-edge-cloud collaborative computing in MEC with task dependencies. Initially, we model inter-task dependencies using Directed Acyclic Graphs (DAG) and propose a task priority queue model to transform the DAG task model into a sequential queue model for easier task scheduling. Subsequently, we formulate a resource-constrained minimization problem for execution delay and energy consumption optimization. To tackle this problem, we introduce a Pruning-based Deep Reinforcement Learning algorithm (PR-DRL) to learn the intricate dependencies between MEC systems and subtasks for optimal offloading decisions. Specifically, PR-DRL incorporates a pruning function that enables the agent to focus on high-probability actions during training while filtering out low-probability actions, thereby achieving rapid algorithm convergence. Simulation results demonstrate that the proposed PR-DRL method outperforms traditional Deep Q Network methods both in terms of convergence speed and offloading performance, surpassing six other baseline task offloading methods.

Keywords: Mobile edge computing; Task offloading; Deep reinforcement learning; Multi-objective optimization.

1. Introduction

With the development of 5G communication, a large number of intelligent mobile applications such as online games, virtual reality (VR), and autonomous driving have experienced rapid growth. These applications are generally resource-intensive and latency-sensitive, making it unrealistic to deploy these applications on resource-constrained mobile end devices (MED). Cloud computing was initially proposed, possessing abundant computing and storage resources to provide on-demand computing services for mobile intelligent applications to alleviate resource shortages in MED. However, cloud computing is distant from MED, resulting in long task processing delays that are intolerable for intelligent applications with strict time requirements.

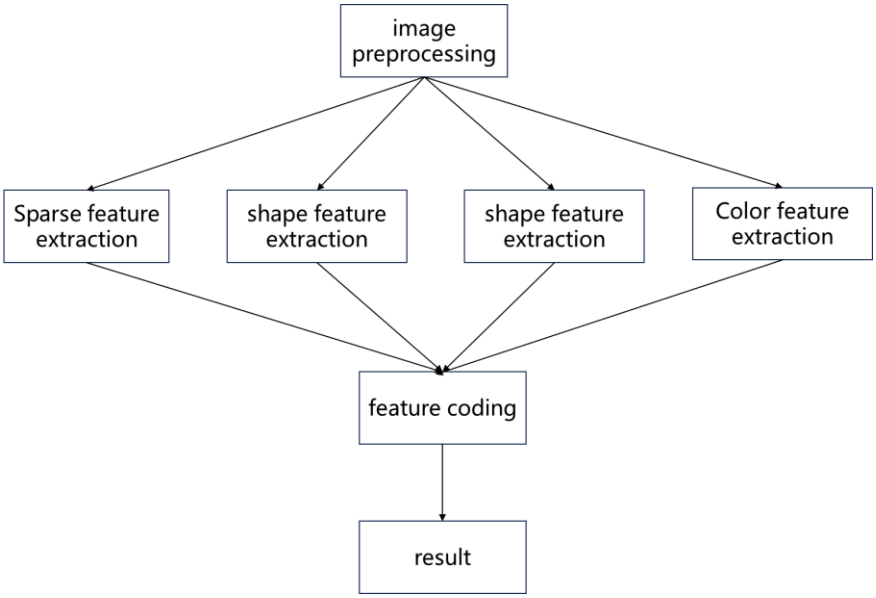

Fig.1 Processing flow for facial recognition

Fortunately, the emergence of edge computing has alleviated this issue. Edge computing extends the computational capabilities of cloud computing to the edge of the network and provides computing services for MEDs, to some extent addressing the significant latency issues caused by tasks originating from the cloud. In 2014, concept of Mobile Edge Computing (MEC) was first proposed by the European Telecommunications Standards Institute. MEC

builds upon edge computing by adding features like mobility and environmental awareness to adapt to the characteristics of mobile applications. The ability of MEC to reduce latency and energy consumption is greatly facilitated by task offloading, which plays a crucial role. Task offloading enhances user Quality of Service (QoS) by determining the timing, quantity, destination, and allocated resources for offloaded tasks. In most mobile intelligent applications, tasks have dependencies on subtasks. An example is illustrated in Figure 1 for facial recognition. For MEC task offloading with task dependencies, it becomes complex as not only the offloading decisions for subtasks need to be determined but also the negative impacts of subtask dependencies on offloading decisions must be addressed. Therefore, designing an effective MEC task offloading method with task dependencies is extremely necessary.

Currently, MEC task offloading methods with task dependencies either excessively rely on system knowledge or exhibit poor algorithm convergence performance. As a result, this paper proposes a pruning function-based Deep Reinforcement Learning (PR-DRL) approach to address task dependency MEC task offloading in end-edge-cloud collaborative computing. The PR-DRL algorithm enhances the convergence performance by introducing pruning functions into the traditional actor-critic framework, enabling the agent to focus on high-probability actions and filter out low-probability actions for faster convergence. The main contributions of this paper are as follows:

(1)Propose an end-edge-cloud collaborative computing MEC system model that effectively expands the computational capacity of MEC servers to avoid edge MEC server overload and resource shortages.

(2)Model task-dependent computing tasks using a Directed Acyclic Graph (DAG) and introduce a priority queue model to convert the DAG task model into a sequential queue model for better handling of these subtasks.

(3)Designing a Pruning Function-based Deep Reinforcement Learning algorithm (PR-DRL) to enhance convergence performance and learn the optimal task offloading strategy for task-dependent edge-cloud collaborative computing MEC.

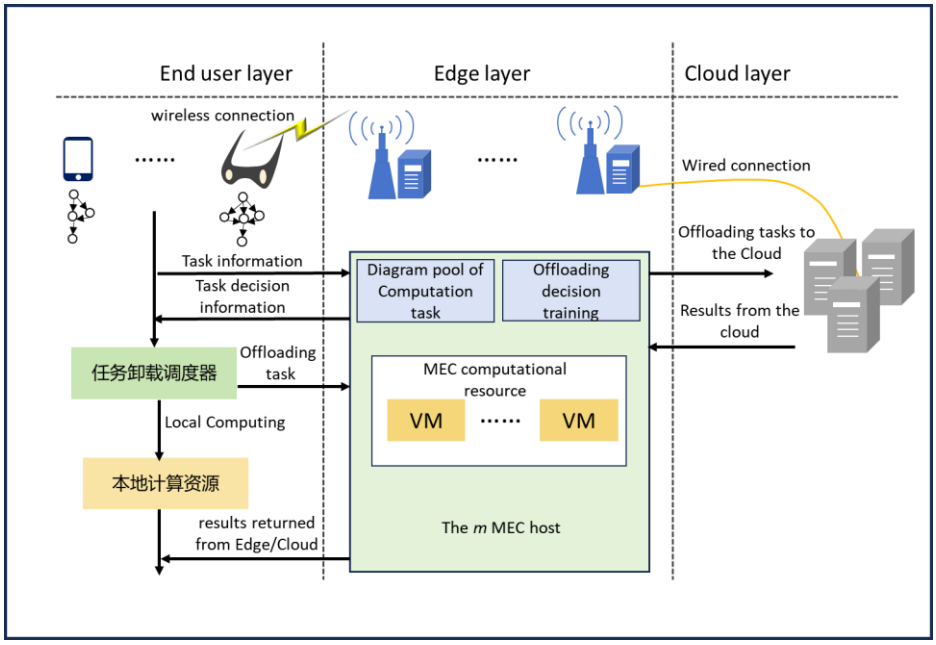

Fig.2 End - edge - cloud collaborative offloading MEC system

2. System Model and Problem Formulation

2.1. System Model

This paper considers a MEC system of three-tier end-edge-cloud collaboration consisting of N end-users, M MEC servers, and a remote central cloud as shown in Figure 2. Each end-user can communicate with MEC edge servers wirelessly, while each edge server can be connected to the remote cloud via fiber optics. When a computation-intensive and latency-sensitive task from an end-user arrives, the end-user sends task information to the MEC server. Let $\mathcal{N} = \{1,2,\ldots,N\}$ represents the set of end-users, $T_n = \{T_{n,1}, T_{n,2}, \ldots, T_{n,K}\}$ denotes the set of subtasks for end-user n, and $\mathcal{M} = \{1,2,\ldots,M\}$ is used to represent the set of MEC edge servers. A subtask Tn can be defined as $T_{n,k} = \{d_{n,k}, \omega_{n,k}, D_{n,k}^{max}\}$, representing the k-th subtask of the n-th end-user, where $d_{n,k}$ is the computational data size of subtask Tn,k, $\omega_{n,k}$ is the computation resource required per bit for the subtask, and $D_{n,k}^{max}$ represents the acceptable time duration to complete task Tn. It is worth noting that due to the inherent parameter attributes of subtasks, these parameters remain unchanged both before and during the offloading process.

The task dependencies of computational task Tn,k can be represented by a directed acyclic graph DAG,denoted as$G_n = (V_n, H_n)$, where $V_n = \{T_{n,i} | \forall i \in \{1,2,\ldots,K\}\}$ is the vertex set of subtasks. The set of edges $H_n = \{e(T_{n,i}, T_{n,j}) | (i,j) \in \{1,2,\ldots,K\} \times \{1,2,\ldots,K\}\}$ representing the relationships between subtasks can be expressed as follows: including e(Tn,i,Tn,,j) indicating the constraints between adjacent nodes Tn,i and Tn,,j of the subtasks. This implies that when the predecessor subtasks not are executed, the subtask $T_{n,i}$ will not be start executed.

Using ST_n to represent the start time of execution of the terminal user's computing task,and WTn,k is used to denote the waiting time of subtask Tn,k, and using FTn,k to represent the completion time of subtask c. Due to the dependency characteristics of the task, we can express the waiting time of subtask Tn,k as:

$$WT_{n,k} = \begin{cases} ST_n, & pre(T_{n,j}) = \emptyset \\ \max\limits_{T_{n,j} \in pre(T_{n,k})} FT_{n,j}, & pre(T_{n,j}) \neq \emptyset \end{cases} \quad (1)$$

In the formula, it is indicated that if subtask Tn,k has no predecessor tasks, it represents the starting node of computational task Tn, and its waiting time is represented by the start time of the computational task STn. Otherwise, when subtask Tn,k is not the starting node of the computational task, its waiting time is represented by the maximum processing completion time among its predecessor nodes.

In order to ensure that subtasks with task dependencies can be successfully executed, this paper adopts a task priority queue model to determine the execution order of different subtasks. The priority of each subtask is related to the latest completion time and the latest execution time. Using LCTn,k to represent the latest completion time of subtask Tn,k, which can be understood as the latest time for the completion of subtask Tn,k, and can be expressed as:

$$LCT_{n,k} = \begin{cases} D_n^{max}, T_{n,k} \in end(T_n) \\ \min\left\{ \min\limits_{T_{n,j} \in suc(T_{n,k})} (LCT_{n,j}, \min\limits_{i \in N \cup M \cup C} T_{n,j}^i), D_{n,k}^{max} \right\}, T_{n,k} \notin end(T_n) \end{cases} \quad (2)$$

The above equation indicates the time delay of subtask Tn,k in the offloading decision, without considering queuing time, which includes user terminal N, MEC server M, and remote cloud C. In order to calculate LCTn,k, iteration must start from the terminal node, denoted as end(Tn). LETn,k is used to represent the latest execution time of subtask Tn,k, which is related to the earliest time LCTn,k when the subtask must be executed, and can be expressed as:

$$LET_{n,k} = LCT_{n,k} - \min_{i \in N \cup M \cup C} ET_{n,k}^i \quad (3)$$

Where $ET_{n,k}^i$ indicates the execution time under the offloading decision i. LETn,k implies the urgency of subtask Tn,k, that is the priority of the subtask. The smaller the LETn,k, the more urgent the subtask Tn,k is. By arranging according to urgency, a task priority queue Q containing all subtasks to be scheduled can be obtained.

End user n can offload a subtask to an MEC server m through a wireless access channel. The task transmission rate from end user n to MEC edge server m, p_n^{tran}represented by the transmission power of the end user, is determined according to Shannon's theorem:

$$R_{n,m} = B_{n,m} \log_2 (1 + \frac{p_n^{tran} \upsilon (\frac{L_{n,m}}{L_0})^{-\mu}}{\sigma^2}) \quad (4)$$

Where $L_{n,m}$ represents the distance between end user n and MEC edge server m, L_0 is the reference distance, μ is the path loss exponent, and υ is the path loss constant, σ^2representing Gaussian channel noise. It is worth noting that the input data for each subtask comes from the output of the direct predecessor subtask and the input data of the corresponding end-user device.

For local execution, the execution delay of subtask Tn,k can be represented as:

$$ET_{n,k}^{local} = \frac{d_{n,k} \omega_{n,k}}{f_n^{local}} \quad (5)$$

Where f_n^{local} is the computing capability of terminal user n . Due to the limited computational resources of terminal devices, this paper assumes that a terminal user can only process one subtask during any given time period. The earliest start time of subtask Tn,k, denoted as $EST_{n,k}^{local}$, represents the earliest start time of the subtask when executed locally, and can be expressed as:

$$EST_{n,k}^{local} = RT_{n,k} + UK_{n,k}^{local} \quad (6)$$

Where $UK_{n,k}^{local}$ represents the available time for locally computing subtasks, which is used to indicate the idle computing resources possessed by the terminal device for local execution. L(n) is used to represent the queue for terminal device n waiting for execution on RTn,k. The available time $UK_{n,k}^{local}$ is determined by the maximum completion time of all remaining subtasks in the queue and can be expressed as:

The completion time of local computation can be expressed as:

$$D_{n,k}^{local} = EST_{n,k}^{local} + ET_{n,k}^{local} \quad (7)$$

The energy consumption can be defined ase $= \varpi f^2$, where ϖ represents parameters related to the chip. Therefore, the expression for energy consumption in locally computing subtask Tn,k is:

$$E_{n,k}^{local} = \varpi d_{n,k} \omega_{n,k} (f_{n,k}^{local})^2 \quad (8)$$

For the task transmission phase, the earliest transmission time $ETT_{n,k,m}^{edge}$ for representing the subtask Tn,k is denoted as:

$$ETT_{n,k,m}^{edge} = RT_{n,k} + UT_{n,k,m}^{edge} \quad (9)$$

Where $UT_{n,k,m}^{edge}$ is the available time from terminal user n to MEC edge server m. In order to ensure that the bandwidth can be properly allocated to the subtask.Using $ERT_{n,k,m}^{edge}$to represent the earliest time for receiving the subtask Tn,k, which is the earliest time for MEC edge server m to receive subtask Tn,kcan be expressed as:

$$ERT_{n,k,m}^{edge} = ETT_{n,k,m}^{edge} + TD_{n,k,m}^{tran} \quad (10)$$

Where $TD_{n,k,m}^{edge}$ represents the time taken for data transmission over the channel, which can be expressed as:

$$TD_{n,k,m}^{tran} = \frac{d_{n,k}}{R_{n,m}} \quad (11)$$

During the edge computing phase, this paper considers that MEC has multi-core computing capabilities. Let$f_{m,u}^{edge}$represent the computing resources of the u-th core of edge server m. Therefore, the computation delay in this sum can be expressed as:

$$ET_{n,k,m}^{edge} = \frac{d_{n,k} \omega_{n,k}}{f_{m,u}^{edge}} + QT_{n,k,m} \quad (12)$$

Where $QT_{n,k,m}$ represents the queuing time of subtask Tn,k, which is determined by whether edge server m is idle. It is worth noting that the number of cores in edge servers will affect their parallel computing capabilities. In order to select a core from edge server m to execute subtask Tn,k, this paper assumes that all available cores have the highest computing power to allocate to tasks, ensuring optimal computational efficiency.

Let R(m) represent the set of subtasks executed on edge server m, ensuring that the number of subtasks in R(m) does not exceed the available cores of edge server m. Therefore, the queuing time can be determined by the minimum completion time $QT_{n,k,m}$ of all subtasks in queue R(m), expressed as:

$$QT_{n,k,m} = \begin{cases} 0, & R(m) = \emptyset \\ \min\limits_{T_{\kappa,j} \in R(m)} FT_{\kappa,j,m}^{edge} - ERT_{n,k,m}^{edge}, & R(m) \neq \emptyset \end{cases} \quad (13)$$

Based on equations (12) - (15), the completion time of subtask Tn,k on the u-th core of edge server m is:

$$D_{n,k,m}^{edge} ERT_{n,k,m}^{edge} + ET_{n,k,m}^{edge} \quad (14)$$

The corresponding energy consumption is:

$$E_{n,k,m}^{edge} = p_n^{tran} TD_{n,k,m}^{tran} + p_n^{idle} TD_{n,k,m}^{idle} \quad (15)$$

Where $TD_{n,k,m}^{idle}$ represents the idle time of terminal user n when not involving wireless data transmission and local computation. It includes waiting time for an idle channel, queuing time of subtasks at the edge, and execution time at the edge. p_n^{idle} corresponds to the power consumption associated with this.

In the wireless transmission stage of cloud computing, the earliest reception time and the earliest transmission time of the subtask are consistent with edge computing. These will not be further elaborated here.

For wired data transmission from the edge to the cloud, let the channel transmission rate R_{ec}between the cloud and the edge be denoted. The transmission time for transferring the subtask Tn,k from the edge to the cloud can be represented as:

$$TD_{n,k}^{ec} = \frac{d_{n,k}}{R_{ec}} \quad (16)$$

For the second cloud computing stage, $f_{n,k}^{cloud}$ representing the computational capability of cloud computing, the processing time of the subtask at the cloud is represented as:

$$ET_{n,k}^{cloud} = \frac{d_{n,k}\omega_{n,k}}{f^{cloud}} \quad (17)$$

According to equations (10)-(13) and (18)-(19), the completion time of offloading the subtask from edge server m to the cloud is represented as:

$$D_{n,k,m}^{cloud} = ERT_{n,k,m}^{edge} + TD_{n,k}^{ec} + ET_{n,k}^{cloud} \quad (18)$$

The corresponding energy consumption of the subtask $T_{n,k}$ at the cloud end is:

$$E_{n,k,m}^{cloude} = p_n^{tran} TD_{n,k,m}^{tran} + p_n^{idle} TD_{n,k,m}^{idle} \quad (19)$$

Here, $TD_{n,k,m}^{idle}$ represents the idle time for end user n apart from data transmission to the edge and local computation. It involves waiting for an idle channel, data transmission time, and execution time at the cloud end.

2.2. Problem Description

Based on the above discussion, the execution latency required for end user n to perform computation task Tn is:

$$D_n = \max_{T_{n,k} \in T_n} D_{n,k} - ST_n \quad (20)$$

Where the execution latency of each subtask Dn,k may depend on the minimum of execution latencies in local terminals, edge terminals, and cloud terminals, which can be represented by the following formula:

$$D_{n,k} = \min\{D_{n,k}^{local}, D_{n,k,m}^{edge}, D_{n,k,m}^{cloud}\} \quad (21)$$

From the above equation, it is evident that the total execution time of computing Tn is related to the longest execution time of a subtask and the starting execution time. The corresponding energy consumption is:

$$E_n = \sum_{k=1}^{K} (x_{n,k} E_{n,k}^{local} + \sum_{m=1}^{M} y_{n,k}^{m} E_{n,k,m}^{edge} + \sum_{m=1}^{M} z_{n,k}^{m} E_{n,k,m}^{cloud}) \quad (22)$$

In the above equation, energy consumption for local computing, edge offloading computing, and cloud offloading computing is simultaneously considered. It is noteworthy that the offloading decisions in each time slot may be higher compared to real-world situations, but this does not affect the formulation and optimization of the offloading optimization problem.

Using a vector $\Omega_{n,k} = [x_{n,k}, y_{n,k}^1 \ldots, y_{n,k}^M, z_{n,k}^1, \ldots, z_{n,k}^M]$ to represent the offloading decisions of subtask Tn,k, wherexn,m denotes local computing for subtask Tn,k, edge server computing, and cloud server offloading for subtask Tn,k, respectively. The offloading strategy should satisfy the following constraints.

Firstly, in order to ensure that a subtask can only be executed in one of local, edge server, or cloud server within a time slot, the following two constraints should be satisfied:

$$x_{n,k}, y_{n,k}^m, z_{n,k}^m \in \{0,1\} \quad (23)$$

$$x_{n,k} + \sum_{m=1}^{M} y_{n,k}^m + \sum_{m=1}^{M} z_{n,k}^m = 1 \quad (24)$$

Secondly, in terms of bandwidth resources. In this study, the time sequence is divided into $\Gamma = \{1,2,\ldots\}$intervals .$B_{n,k,m}^t$ represents the bandwidth possessed by a subtask within interval t, assuming $B_{n,k,m}^t$ remains constant throughout any interval. Furthermore, the total bandwidth resources possessed by all subtasks within any interval should not exceed the available maximum bandwidth resources. The bandwidth constraints can be expressed as:

$$\sum_{n=1}^{N} \sum_{k=1}^{K} B_{n,k,m}^t \le B_{max}, \forall t \in \Gamma, m \in M \quad (25)$$

Finally, the number of cores on the MEC server should also be restricted to represent the cores used on edge server m within time interval t, which should not exceed the maximum number of cores available in any given interval. This can be expressed as:

$$P_m^t \le \Delta, \forall t \in \Gamma, m \in M \quad (26)$$

The offloading decisions for all tasks in the system are represented using vectors $\mathcal{E} = [\Omega_{1,1}, \Omega_{1,2}, \ldots, \Omega_{N,K}]$. Considering task dependencies and resource constraints, the mathematical expression for the multi-objective optimization problem can be defined as:

$$\min_{\mathcal{E}} \frac{1}{N} \sum_{n=1}^{N} (D_n + E_n) \quad (26)$$

st. C1: $FT_{n,k} \le D_{n,k}^{max}, \forall n \in N, k \in K$

(1), (25)-(26), (27)-(28)

3. Pruning function - deep reinforcement learning for task offlaoding

3.1. MDP model of the original problem

Key Three Elements in transforming the original problem into Markov Decision Process (MDP), namely state space S, action space A, and reward function R, are elaborated on in detail below.

State Space: In this study, the system's state space consists of two parts, namely subtask states and global states. Subtask states include local computional resources, attributes of the subtasks, and distances between end-users and base stations. . Within a decision time period (t), the decision algorithm makes decisions based on the current subtask states and global states. This state space can be defined as:

$$S = \{s_t | s_t = \{T_{n,k}, L_{n,m}, f, s_{wc}, s_{edge}\}\} \quad (27)$$

Where the attributes Tn,k of a subtask include the workload of the subtask, the number of CPU cycles per bit required to execute the subtask, and the deadline for executing the subtask.frepresents computing resources across local, edge servers, and cloud servers. The state information s_{wc} of the wireless channel is composed of the queue of tasks waiting to

be offloaded to the corresponding edge server and the remaining bandwidth. The state information s_{edge} of an edge server consists of the number of tasks queued for execution on the respective edge server and the remaining idle cores on that edge server.

Action Space: Within a decision time period t, only one subtask can be scheduled. Therefore, the action space includes decisions on local execution, edge server execution, and offloading to cloud execution. Thus, an action space can be represented as:

$$A = \{a_t | a_t \in \{0,1,\ldots,2M\}\} \quad (28)$$

$a_t = 0$ means the subtask is executed locally, $a_t \in \{1,2,\ldots,M\}$means the subtask is offloaded for execution on an MEC edge server, and $a_t \in \{M+1, M+2,\ldots,2M\}$ represents the scenario where the subtask is further offloaded by edge server m for processing in the cloud.

Reward Function: The reward function affects the decisions of the agent, and the quality of the reward function design will impact the final actions, i.e., offloading decisions. Therefore, the reasonable design of the reward function is particularly important. Using Dn,k and En,k to represent the execution delay and energy consumption of subtask Tn,k within a certain offloading decision period, the normalized expression for the reward function is:

$$R(s_t, a_t) = (1-\rho)\frac{D_{n,k}^{local} - D_{n,k}}{D_{n,k}^{local}} + \rho\frac{E_{n,k}^{local} - E_{n,k}}{E_{n,k}^{local}} \quad (29)$$

Where ρis a weighting factor used to adjust the trade-off between execution delay and energy consumption, satisfying $\rho \in [0,1]$. The above expression can be understood as follows: when the subtask is computed locally, the reward is 0, and the agent does not receive a reward; however, when offloading computation occurs, the reward is greater than zero, indicating a reasonable offloading decision.

3.2. Algorithm design

In this study, the Proximal Policy Optimization (PPO) algorithm proposed in reference is borrowed to address the MEC task offloading decision problem involving task-dependent edge-cloud collaboration. The PPO algorithm also includes an actor-critic network framework, where the actor network determines efficient actions in the action set and learns the policy through gradient descent to maximize rewards. On the other hand, the critic network estimates the rewards from the actor by minimizing the loss function.

Compared to other policy gradient algorithms, PPO constrains the update of network parameters within a trusted region by using a clipping function. This feature enables the PPO algorithm to have more stable training and better training results. For each episode, the agent collects state information about tasks and the system's global state, then makes actions based on the policy, such as task offloading decisions, until the task queue is empty. Finally, the agent trains the network parameters based on the loss function.

The PPO algorithm is based on the state value V and approximates the value function using a deep neural network DNN. The value function $V_\theta(s_t)$ is used to approximate the negative of the cost function $J(s_t)$. The objective function of the critic network is to minimize the difference between $V_\theta(s_t)$ and $-J(s_t)$, which can be expressed as:

$$J^V(\theta) = \mathbb{E}_t[(V_\theta(s_t) - (-J(s_t)))^2] = \mathbb{E}_t[J_t^V(\theta)] \quad (30)$$

Where $J^V(\theta)$ is the optimal cost function, and $\mathbb{E}_t[J_t^V(\theta)]$ represents the empirical mean of a finite batch of samples obtained during the alternating process of sampling and optimization.

To make the actions taken by the agent more targeted, in other words, to automatically filter out some low-probability actions, this study adopts the PPO algorithm with a pruning function and an attached action mask. In the PPO method with action masks, the environment must provide a mask indicating the availability of actions, and the agent must normalize the probabilities of available actions based on the obtained mask, setting the probability of unavailable actions to zero.

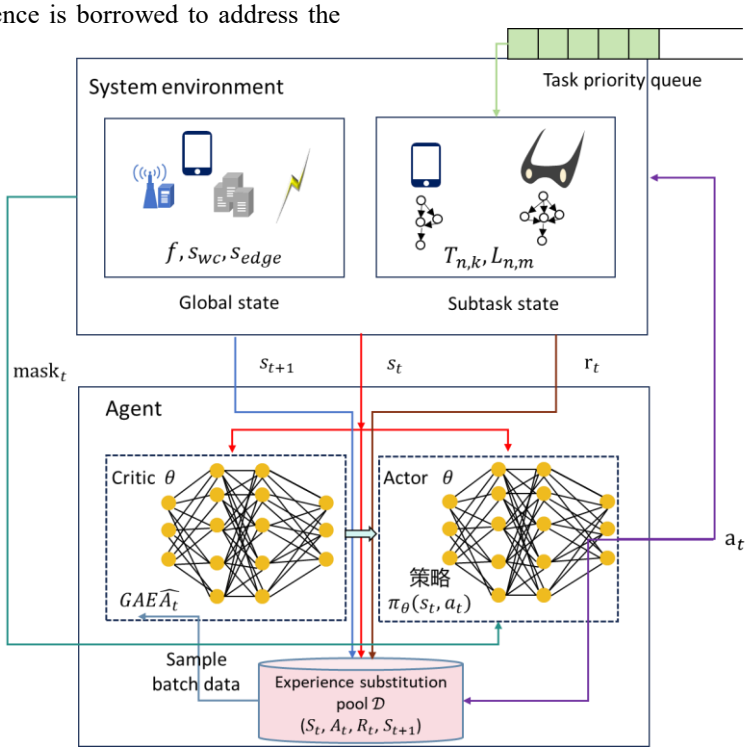

Fig.3 PR-DRL algorithm frame

Similar to reference, the training objective is to maximize the pruning agent's objective function:

Algorithm 1 Task offloading based on PR-DRL

Input: f_n^{local} , $f_{n,m}^{edge}$, f^{cloud} , $\forall n \in N, k \in K, m \in M, d_{n,k}, \omega_{n,k}, D_{n,\ k}^{max}, L_{n,m}, p_n^{tran}, p_n^{idle}$

$\chi_1, \chi_2, \gamma, \alpha, \epsilon, B_{max}, \sigma^2, \nu, \mu, Batch;$

Output:Offloading decision $x_{n,k}, y_{n,k}^1 \ldots, y_{n,k}^M, z_{n,k}^1, \ldots, z_{n,k}^M$

1:Initialize the experiential replacement pool $\mathcal{D}$ and neural network parameters θ;

2: **for** each episode in episodes **do**

3: Reset the environment to get the initial state s_0 ;

4: Initialize the task priority queue Q;

5: **for** each subtask $T_{n,k}$ **do**

6: The $LCT_{n,k}$ and $LET_{n,k}$ of each subtask are calculated according to equations (2) and (3) respectively;

7: $LET_{n,k}$ is used to sort the queue subtasks in ascending order;

8: **end for**

9: **for** The task priority queue Q is not empty **do**

10: Select the highest priority subtask from the queue Q;

11: Gets the current status s_t and the action $mask_t$ m from the environment;

12: Select Action a_t under policy π_θ based on $mask_t$;

13: Execuate Action a_t and get reward $r_t(s_t, a_t)$;

14: Transition from current state s_t to next state s_{t+1};

15: Store (s_t, a_t, r_t, s_{t+1}) into $\mathcal{D}$;

16: Execuate next step $t \leftarrow t + 1$;

17: **end for**

18: The advantage estimator $\widehat{A_1}, \ldots, \widehat{A_T}$ are calculated according to equation (35);

19: **for** each epoch **do**

20: Randomly sample $Batch$ of data from $\mathcal{D}$;

21: Update network parameter θ according to equation (38);

22: **end for**

23: The old DNN parameter θ_{old} is replaced by a new DNN parameter θ, and the old strategy $\pi_{\theta_{old}}$ is updated. Finally, the probability ratio $p_t(\theta)$ is calculated.

24: **end for**

$$J^{\text{clip}}(\theta) = \mathbb{E}_t[\min(p_t(\theta)\widehat{A_\theta}, \text{clip}(p_t(\theta), 1-\epsilon, 1+\epsilon)\widehat{A_\theta}) \quad (31)$$

Where θ represents the shared parameters of the actor network and critic network; $\mathbb{E}_t$ is the expectation over the sampled batch data; $p_t(\theta)$ denotes the probability ratio, given by $p_t(\theta) = \pi_\theta(a_t|s_t)/\pi_{\theta_{\text{old}}}(a_t|s_t)$, with π_θ and $\pi_{\theta_{\text{old}}}$ representing the learning new policy and old policy, respectively. The introduction of the probability ratio is mainly to avoid excessively updating the old parameters and focus more on updating the new parameters. ϵ is a hyperparameter controlling the pruning scope; $\text{clip}(\cdot)$ is the pruning function. The pruning function restricts the probability ratio within the range $[1-\epsilon, 1+\epsilon]$ by setting hyperparameters. It serves as a Generalized Advantage Estimator (GAE) [], which, as an advantage function estimator, effectively reduces the error in gradient estimation and $\widehat{A_\theta}$ can be expressed as:

$\widehat{A_\theta} = \sum_{i=0}^{T-t}(\gamma\lambda)^i\delta_{t+i}$, Where $\delta_{t+i} = -r_t(s_t, a_t) + \gamma V_\theta(s_{t+1}) - V_\theta(s_t)$ is the residual of the state value function V_θ(approximated by a critic network), and $\gamma$is the discount factor used to balance bias and variance.

Additionally, an entropy gain term is added to the loss function, allowing the model to explore the environment fully during training. Combining the factors mentioned above with equations (32) and (33), the loss function can be expressed as:

$$J^{PPO}(\theta) = \mathbb{E}_t[J^{clip}(\theta) - \chi_1 J_t^V(\theta) + \chi_2 \mathbb{S}_{\pi_\theta}(s_t)] \quad (32)$$

Where χ_1 and χ_2 are respectively the coefficients for squared error loss and entropy gain, primarily serving as regulators, with $\mathbb{S}$ being the entropy gain.

4. Simulation experiment and performance evaluation

4.1. Setting of experimental parameters and the selection of benchmark algorithm

Table.1 Default main parameters

Description	Value
α	0.0004
γ	0.9
ϵ	0.3
$B_{n,m}$	[3,6]MHz
B_{max}	80MHz
f_n^{local}	[1.2,2] GHz
f_m^{edge}	[2.7,5] GHz
f^{cloud}	6 GHz
p^{tran}	1.3
p^{idle}	0.2
$d_{n,k}$	[500,1000] KB
$\omega_{n,k}$	40 Cysle/Bit
R_{ec}	5 MB/s
σ^2	10^{-9}W
κ	10^{-27}

In the simulation experiment, this study sets the radius of the signal coverage area with each MEC server's base station to 150m. This means that each terminal user running a computing task can communicate with the base station as long as they are within 150m of it. The default configuration includes four edge servers and 25 terminal users. The positions of terminal users are randomly generated around MEC base stations using a Python simulator. A cloud server communication range of 10km is set up in the experiment. Terminal users' tasks are generated by a Python simulator [30], creating different types of DAG task sets, which can be controlled by four parameters: the number of nodes, parameters controlling DAG length and height, task scale, and

edges. Performance validation in this study also uses Python 3.8.0 as the language tool, Pytorch 1.5.0 to implement a DNN, with an ASUS ROG-STRIX-GTX 1080Ti-O11G-GAMING hardware platform and 16GB of RAM. The DNN consists of three fully connected layers with 128 units, and the activation function employed is Tanh. The other parameters are referenced from literature [31] and [32], as shown in Table 1.

The following six algorithms were selected as benchmark algorithms to evaluate the performance of the PR-DRL algorithm:

Local computation (LC): All subtasks are executed on local terminal devices.

Edge-first (EF): All subtasks are offloaded and executed on edge servers.

Cloud-first (CF): All subtasks are offloaded and executed on cloud servers.

Random offloading strategy (ROS): The offloading decision for each subtask is random.

Genetic Algorithm (GA): For the genetic algorithm, this study selected reference [10] for comparison.

DQN-based Algorithm (DQN): As mentioned in Chapter 2, the DQN algorithm based on value iteration is also chosen as a benchmark algorithm for comparison in this study.

4.2. Experimental results and analysis

First, evaluate the convergence performance of the PR-DRL algorithm proposed in this study. The intelligent agent is trained for 1300 episodes with a delay weight value of 0.3. From Fig 4, it can be observed that both the PR-DRL algorithm proposed in this study and the DQN method show an increase in average reward as the number of episodes increases during training. This indicates that the offloading strategy is gradually learning the system environment and task states. Additionally, it is evident from the graph that the PR-DRL algorithm converges faster than DQN and achieves higher rewards. After convergence, the PR-DRL algorithm remains relatively stable, while DQN fluctuates. This is because the PR-DRL algorithm's pruning function, action-masking, and Adaptive Greedy Exploration (AGE) mechanism focus the intelligent agent on high-quality actions and ignore inferior ones, leading to high-quality and fast convergence performance.

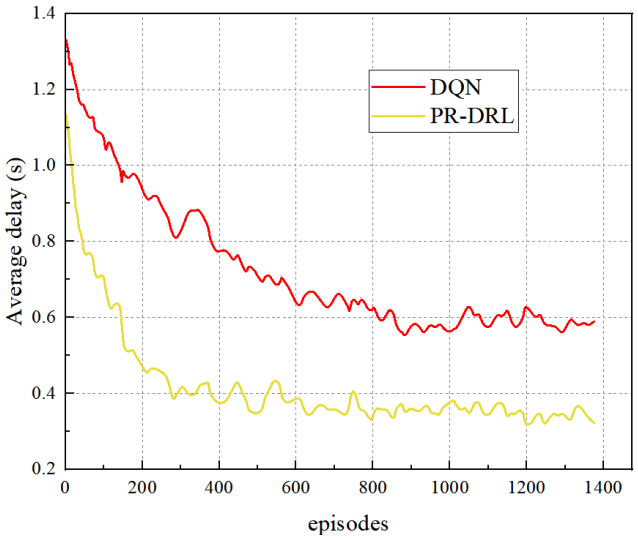

Fig.4 Average reward of convergence of the algorithm

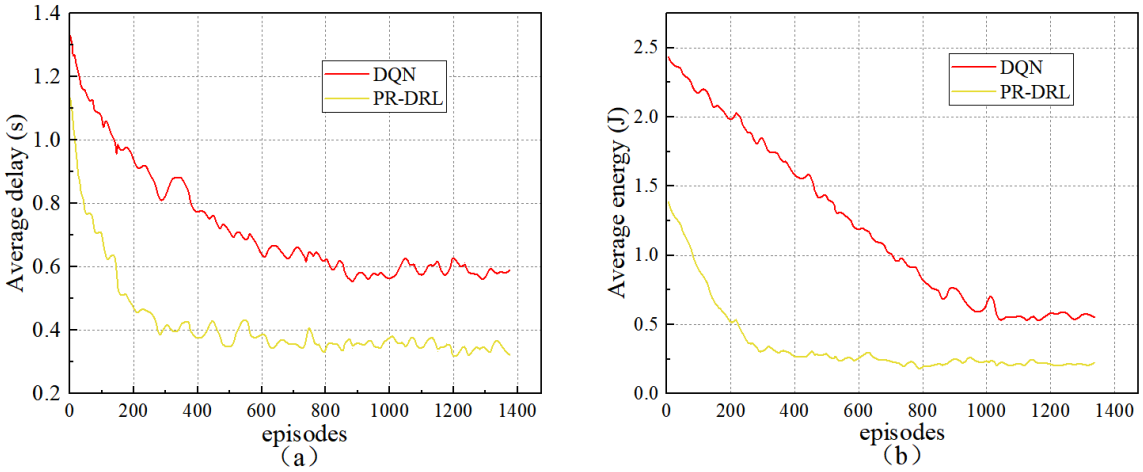

Fig.5 Convergence performance of the algorithm in terms of average reward: (a) average delay, (b) average energy consumption.

From equation (31), it can be inferred that the value of the reward function is influenced by both the calculation delay and energy consumption. The changes in average delay and average energy consumption corresponding to this during the learning process are illustrated in Figure 5. For the PR-DRL and DQN deep reinforcement learning strategies, both the execution delay and energy consumption decrease as the number of training iterations increases. Around the 700th episode mark, the PR-DRL algorithm reaches its minimum for both average delay and average energy consumption,

while the DQN algorithm takes until around the 1000th episode to reach its minimum. Therefore, from the perspective of average delay and energy consumption, it can be observed that the proposed method in this study effectively minimizes the optimization objectives. Compared to traditional DQN, the optimized values are smaller with faster convergence performance.

Figure 6 displays the offloading performance of different algorithms across various subtasks. Figures 6 (a) and 6 (b) respectively show the average delay and average energy consumption for all end-users. It can be observed from the graphs that as the number of subtasks increases, the delays and energy consumption of all offloading strategies also increase. This is due to the fact that with a higher number of subtasks for each computational task, the MEC system needs to handle more subtasks, leading to tighter computational resource constraints and increased queuing times. Consequently, this results in greater delays and energy consumption for the computational tasks.For strategies prioritizing edge execution, as the number of subtasks exceeds 15, their performance deteriorates due to computational resource limitations, unlike cloud execution, which does not face such constraints. Furthermore, compared to the other six offloading strategies, PR-DRL exhibits slower growth in delay and energy consumption as the number of subtasks increases. Therefore, the PR-DRL offloading strategy, equipped with pruning functions, can focus on more critical actions in scenarios with numerous subtasks, thereby achieving superior performance.

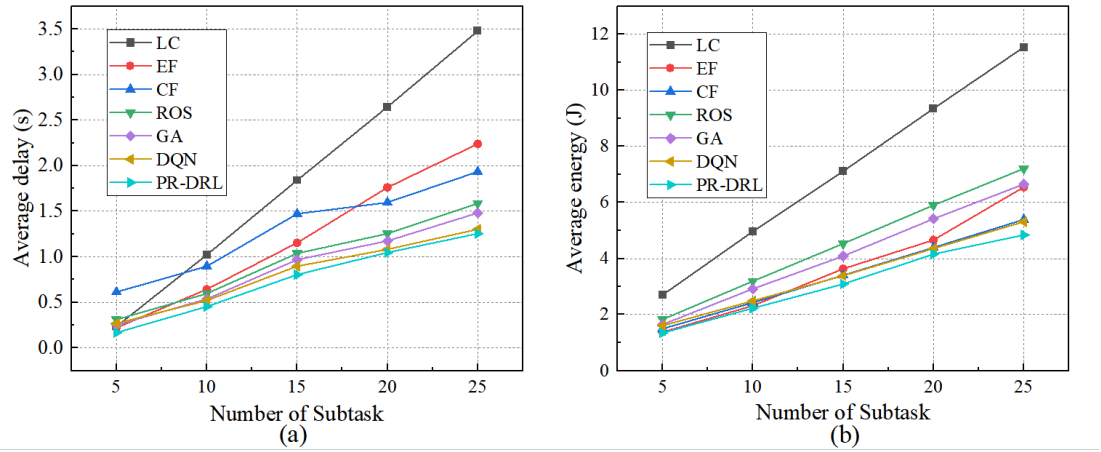

Fig.6 Impact of different numbers of subtasks on offloading performance: (a) average delay, (b) average energy consumption.

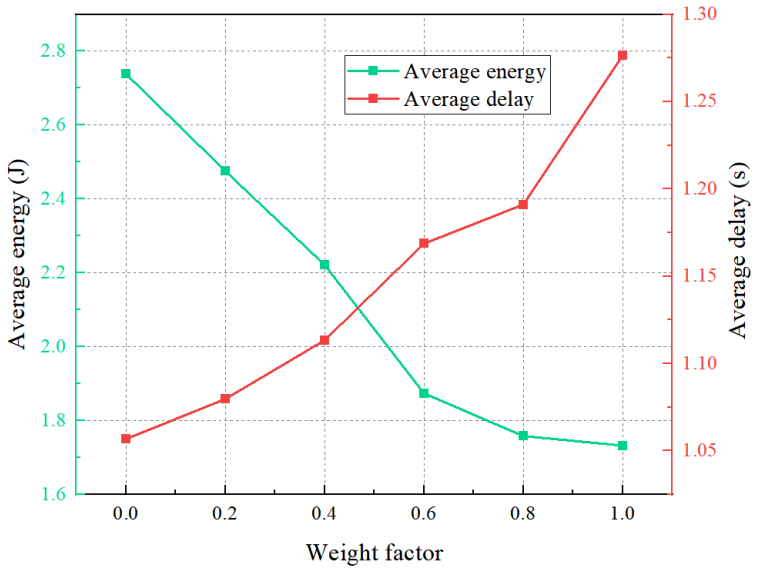

Fig.7 Impact of weight adjustment factor ρ on offloading performance

Fig 7 illustrates the impact of adjusting weighting factor ρ for execution delay and energy consumption on MEC task offloading strategies. From the graph, it can be observed that when the weighting factor is small, emphasizing lower execution delays results in lower average delays but higher energy consumption. As the weighting factor increases, the proportion of energy consumption also increases, causing the PR-DRL task offloading strategy to focus more on energy consumption. Consequently, end-user energy consumption decreases while delays increase. By adjusting the weighting factors, the algorithm proposed in this paper can better balance average energy consumption and execution delay.

5. Conclusion

In this article, the problem of task offloading in Mobile Edge Computing (MEC) with end-edge-cloud collaborative computing is considered, where task dependencies are represented using Directed Acyclic Graphs (DAG). To

address this issue, we first propose a three-tier MEC system architecture for multiple users and multiple MEC servers, allowing computational tasks to be executed at the user side, edge side, or cloud side. Secondly, we utilize a task priority queue to transform the task model of the DAG into a sequentially arranged queue model based on urgency. Subsequently, we introduce a Deep Reinforcement Learning algorithm based on pruning functions (PR-DRL) to learn optimal offloading strategies by understanding complex system and task states. Simulation results demonstrate that the proposed PR-DRL method not only achieves the lowest execution latency and energy consumption overhead but also exhibits faster convergence performance.

References

[1] Nadir, Zinelaabidine, et al. Immersive services over 5G and beyond mobile systems. IEEE Network 35.6 (2021): 299-306.

[2] Antonopoulos, Nick, and Lee Gillam. Cloud computing. Vol. 51. No. 7. London: Springer, 2010.

[3] Mao, Yuyi, et al. A survey on mobile edge computing: The communication perspective. IEEE communications surveys & tutorials 19.4 (2017): 2322-2358.

[4] Carvalho, Gonçalo, et al. Edge computing: current trends, research challenges and future directions. Computing 103.5 (2021): 993-1023.

[5] Hu, Yun Chao, et al. Mobile edge computing—A key technology towards 5G. ETSI white paper 11.11 (2015): 1-16.

[6] Qin, Meng, et al. Service-oriented energy-latency tradeoff for IoT task partial offloading in MEC-enhanced multi-RAT networks. IEEE Internet of Things Journal 8.3 (2020): 1896-1907.

[7] Yu, Wenmeng, and Hua Xu. Co-attentive multi-task convolutional neural network for facial expression recognition. Pattern Recognition 123 (2022): 108401.

[8] Schulman, John, et al. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347 (2017).

[9] Tang, Cheng-Yen, et al. Implementing action mask in proximal policy optimization (PPO) algorithm.ICT Express 6.3 (2020): 200-203.

[10] Schulman, John, et al. High-dimensional continuous control using generalized advantage estimation. arXiv preprint arXiv:1506.02438 (2015).