Complex & Intelligent Systems (2024) 10:3283–3304
https://doi.org/10.1007/s40747-023-01322-x

ORIGINAL ARTICLE

DRL-based dependent task offloading with delay-energy tradeoff in medical image edge computing

Qi Liu[1,2] · Zhao Tian[3] · Ning Wang[1,2] · Yusong Lin[2,3,4]

Received: 15 June 2023 / Accepted: 13 December 2023 / Published online: 29 January 2024

Abstract

Task offloading solves the problem that the computing resources of terminal devices in hospitals are limited by offloading massive radiomics-based medical image diagnosis model (RIDM) tasks to edge servers (ESs). However, sequential offloading decision-making is NP-hard. Representing the dependencies of tasks and developing collaborative computing between ESs have become challenges. In addition, model-free deep reinforcement learning (DRL) has poor sample efficiency and brittleness to hyperparameters. To address these challenges, we propose a distributed collaborative dependent task offloading strategy based on DRL (DCDO-DRL). The objective is to maximize the utility of RIDM tasks, which is a weighted sum of the delay and energy consumption generated by execution. The dependencies of the RIDM task are modeled as a directed acyclic graph (DAG). The sequence prediction of the S2S neural network is adopted to represent the offloading decision process within the DAG. Next, a distributed collaborative processing algorithm is designed on the edge layer to further improve run efficiency. Finally, the DCDO-DRL strategy follows the discrete soft actor-critic method to improve the robustness of the S2S neural network. The numerical results prove the convergence and statistical superiority of the DCDO-DRL strategy. Compared with other algorithms, the DCDO-DRL strategy improves the execution utility of the RIDM task by at least 23.07, 12.77, and 8.51% in the three scenarios.

Keywords Task offloading · Medical image cloud · Deep reinforcement learning · Directed acyclic graph · Soft actor-critic

Introduction

Currently, the amount of image data, which exceeds 34 trillion GB, imposes a heavy workload on doctors [1]. The radiomics-based image diagnosis model (RIDM) [2] is a time-consuming and computation-intensive (CI) mature clinical diagnostic method. As a commonly used solution for hospitals to handle large-scale computing tasks, the medical image cloud [3, 4] is far from the hospital TD, resulting in significant transmission delay and energy consumption (DEC).

Task offloading (TO) [5] as a critical technology of edge computing (EC) [6] offers a solution to the above dilemma by offloading the CI task to a closer edge server (ES). This can effectively decrease delay but also increase energy consumption. Thus, choosing an appropriate offloading strategy for the RIDM task to trade off DEC is a key problem [7]. In fact, the complexity of medical image data requires significant computational resources to support the execution of various phases in RIDM. In addition, the combination of various methods during each radiomics phase results in different RIDM tasks. Hence, we believe that a good TO strategy should improve the RIDM run efficiency and adapt to different RIDM environments. Nevertheless, to obtain such a TO strategy, the following issues in the RIDM task should be addressed.

✉ Yusong Lin
yslin@ha.edu.cn

Qi Liu
qliu@gs.zzu.edu.cn

Zhao Tian
tianzhao@zzu.edu.cn

Ning Wang
wning@ha.edu.cn

1 School of Computer and Artificial Intelligence, Zhengzhou University, Zhengzhou 450001, China

2 Collaborative Innovation Center for Internet Healthcare, Zhengzhou University, Zhengzhou 450052, China

3 School of Cyber Science and Engineering, Zhengzhou University, Zhengzhou 450002, China

4 Hanwei IoT Institute, Zhengzhou University, Zhengzhou 450002, China

The run efficiency of the RIDM task is constrained as existing TO solutions are divided into binary and partial solutions based on task separability [8]. However, considering the complexity of the radiomics workflow, this solution of a simplistic partition task is proven unsuitable. Therefore, it is crucial to correctly partition and handle subtasks with multiple dependencies based on the internal logic of the workflow for the successful execution of RIDM tasks [9]. On the other hand, due to limited resources in ES, executing assigned subtasks independently results in slower speeds, thereby impacting user experience [10]. Thus, developing collaboration between ESs is necessary to speed up task processing.

The TO problem mentioned above is NP-hard [11]. Many solutions using heuristic [8] or approximation [12] (HA) algorithms have been developed. Nevertheless, to adapt to different RIDM environments, it is impractical to apply HA algorithms that depend on expert knowledge and precise mathematical models (EKM). Model-free deep reinforcement learning (DRL) [13, 14] has received widespread attention due to not relying on manual intervention or EKM. However, DRL suffers from lower sampling efficiency and brittleness to hyperparameters [15]. Hence, there is an urgent need to choose a robust DRL algorithm for the RIDM-TO problem to adapt to different RIDM environments.

Therefore, the following three challenges need to be solved in RIDM task offloading. First, how can represent the dependencies between modules (i.e., subtasks) in the RIDM task? Second, how can collaborative computing between ESs be explored to improve the efficiency of RIDM task execution? How can the drawbacks of model-free DRL be overcome to improve the robustness of the offloading decision-making process?

Motivated by the above challenges, we propose a distributed collaboration-dependent task offloading strategy based on DRL (DCDO-DRL). In particular, considering the uniqueness of the radiomics workflow and the limited resources of ESs, we combine reinforcement learning (RL) [16], a sequence-to-sequence (S2S) neural network [17] and EC to optimize the offloading decision-making process of the RIDM task. Specifically, the main contributions of this article are summarized as follows.

1. This article proposes a DCDO-DRL strategy that can improve the RIDM execution efficiency and adapt to different RIDM environments. In DCDO-DRL, we use RL to model the TO problem as a Markov decision process (MDP). DCDO-DRL aims to maximize the RIDM task utility, a weighted sum of DEC generated by execution.
2. In a radiomics workflow-based medical scenario, the RIDM task consists of several dependent subtasks that can be modeled as a directed acyclic graph (DAG). The offloading decision process in the DAG is represented by the sequence prediction of the S2S neural network. A multiple ESs distributed collaboration processing (DCP) algorithm based on the network topology and the resources is proposed for offloading subtasks to the edge.
3. The DCDO-DRL strategy utilizes a discrete soft actor-critic (SAC) method based on maximum entropy to learn a robust DRL algorithm empowered by the S2S neural network, enabling it to adapt to different RIDM environments. In particular, we modify the action space of the SAC algorithm from continuous to discrete to adapt to the offloading actions in the RIDM task.
4. We prove the convergence and statistical superiority of the DCDO-DRL strategy. The numerical results reveal that, compared with other algorithms, the DCDO-DRL strategy improves the execution utility of the RIDM task by at least 23.07, 12.77, and 8.51% in the three scenarios.

Related work

The massive amount of data poses challenges to traditional medical image processing using MapReduce and Hadoop [18–21]. Cloud Computing is a proven way to manage and process big data [22, 23]. However, there are great distances between CC and TDs in the medical imaging cloud. Transferring a large amount of image data will incur a significant delay and energy consumption. Task offloading has attracted wide attention as one of the most promising solutions to the above issue [24]. Unfortunately, researchers have paid less attention to improving medical image processing by task offloading, mainly in fields such as the Internet of Vehicles and unmanned aerial vehicles.

The existing task offloading strategies are divided into two categories: HA-based TO and DRL-based TO. HA-based TO strategies are achieved through expert knowledge or precise mathematical models. For example, Li et al. proposed a binary offloading policy based on an alternating direction method of a multiplier algorithm to achieve power minimization [25]. To minimize system cost, Pan et al. proposed a heuristic algorithm to solve the binary computation offloading problem, which is formulated as a mixed-integer non-linear programming problem [26]. Chen and Wang proposed a situation-aware binary offloading strategy based on heuristic algorithms that maximize delay and energy consumption by opportunistically adopting changing resource availability conditions [8]. Zhang et al. To minimize task latency and energy consumption, A proposes an offloading scheme that adjusts the task priority in the subtask dependency graph [27]. Fu et al. aimed to minimize energy consumption during task execution by an iterative algorithm based on successive convex approximation [28]. Bi et al. incorporated PSO and Genetic Learning, designing

a meta-heuristic algorithm to minimize the system energy [29]. These above studies adopt HA algorithms endorsed by expert knowledge, which are difficult to adjust dynamically according to the different environments. In addition, when the scale of the task offloading problem is large, the decision generation time is very long and only an approximate optimal solution can be obtained..

For DRL-based strategies, continuously optimize offloading decisions through online learning and gradually get the optimal offloading strategy. For example, Wang et al. incorporated Lyapunov Optimization, Multi-armed Bandit, and Extreme Value Theory, proposing a learning-based energy-aware task offloading policy [30]. Seid et al. formulated the task offloading problem as a Markov decision process considering a stochastic game, to minimize energy consumption and delay [31]. Similarly, Alam and Jamalipour modeled the task offloading problem as a Stochastic Game optimization problem and solved it with a multi-agent DRL-based Hungarian algorithm [32]. Zhan et al. proposed a policy gradient-based DRL approach to solving the task offloading problem, which is formulated as a partially observable Markov decision process [33]. Chen et al. considered the task's relevance and designed a distributed DRL algorithm to solve the task offloading in industrial networks [34]. Some researchers combine blockchain and DRL to solve the task offloading problem. Wang et al. formulated the task offloading problem as a Markov game and combined Blockchain, DRL and Mean Field Theory to propose a secure learning-based off-chain task offloading algorithm [35]. To guarantee the security and reliability of task offloading, Shi et al. incorporated a DRL-based computational offloading scheme and a consensus algorithm based on practical Byzantine fault tolerance (PBFT) in the smart contract of blockchain [36]. The model-free DRL frameworks [37] used above, such as deep Q-learning, PPO, and DDPG, have although self-learning and adaptive characteristics, suffer from poor sample efficiency and hyperparameter brittleness. There is an urgent need to choose a robust DRL algorithm for the RIDM-TO problem to adapt to different RIDM environments.

All the above solutions assume that the task is dependent and has no internal dependencies. However, most tasks in real life are not like this, especially the RIDM task. If dependencies are ignored when making offloading decisions, it will reduce strategy performance. Furthermore, these solutions also fail to consider the limited computing resources of edge servers. This resource condition makes it difficult to undertake computation-intensive tasks like RIDM. Therefore, for the RIDM task offloading scenario, this article proposes a DCDO-DRL strategy, which is designed to maximize the execution utility of the RIDM task. We propose a DCP algorithm to develop collaborative computing between ESs. We adopt DAG to represent the dependencies of the RIDM task. The offloading decision process in the DAG is represented by the sequence prediction of the S2S neural network. Finally, to obtain a robust offloading strategy, the DCDO-DRL strategy utilizes discrete SAC to train the S2S neural network.

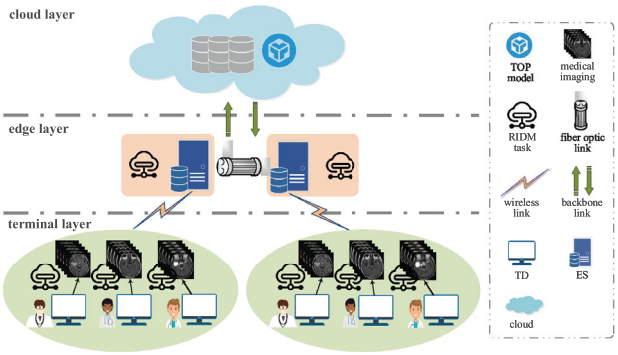


Fig. 1 Three-layer hierarchical system framework

System model and problem formulation

This section presents first a hierarchical system architecture. Next, convert the radiomics workflow into a DAG to demonstrate the dependencies of the RIDM task. Then, the computation and transmission process is described in the local and edge layers. Finally, a utility function is designed to formalize the goal of this article.

System model

As illustrated in Fig. 1, we consider a three-layer hierarchical system framework with terminal-edge-cloud collaboration for RIDM task execution. This system comprises multiple terminal devices, multiple edge servers, and a centralized cloud. TDs are endowed with limited computation and storage capabilities, typically for performing small-scale RIDM tasks in hospital PCs. ESs are endowed with large computation and storage capabilities. The communication between ESs and TDs within the communication range communicate, between ESs and between CC and ESs is carried out via a wireless link, fiber optic link and backbone link, respectively. The CC has near-infinite resources to afford the computation and storage capabilities to train a task offloading planner (TOP) model. TD and ES execute tasks based on the task positions assigned by the cloud-trained TOP model. For readability, Table 1 summarizes the notation used in this article. To clearly explain the hierarchical framework for RIDM task execution, it is formalized as follows:

Definition 1 RIDM task offloading system model is a 12-tuple: RIDM-TO $= (S, D, \text{DCP}, G, B, \mu, \phi, \zeta, V^l, V^s, T^{total}, \Psi^{total})$, which is described in Appendix.

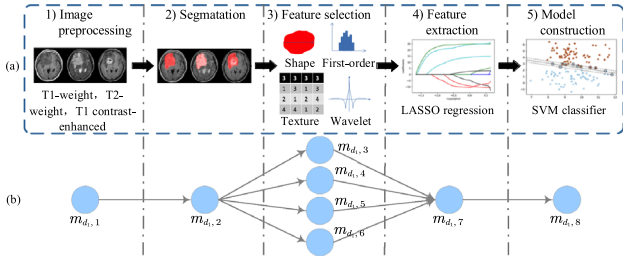


Fig. 2 An example of a radiomics-based prediction of diffuse glioma grading model with DAG conversion

Conversion of radiomics workflow to DAG

The RIDM task is often treated holistically in studies, neglecting the internal dependencies. It may lead to a negative impact on the DEC of the task execution. Thus, we design a more fine-grained division of the RIDM task according to the radiomics workflow.

The radiomics workflow consists of multiple interdependent modules. A simple workflow has linear dependencies between modules. However, complex workflow involves more complicated internal dependencies between modules. Each module can be seen as a medical subtask. We modeled RIDM as different DAGs based on the selection of the method in the actual radiomics workflow. To clearly explain the DAG topology of the RIDM task, it is formalized as follows:

Definition 2 DAG topology of RIDM task model is 2-tuple: $g_{d_i} = (M, Z)$, where $M = \{m_{d_i,v} | v = 1, 2, \ldots, V\}$ is the vertex finite set that represents medical subtasks. $Z = \{\vec{z} < m_{d_i,v}, m_{d_i,w} > | v, w \in T\}$ is the directed edge finite set that expresses the dependencies among medical subtask. $m_{d_i,v}$ is the immediate predecessor medical subtask of $m_{d_i,w}$.

Figure 2 shows the workflow and DAG of the diffuse glioma grading (DGG) prediction model based on radiomics in [38]. The radiomics workflow presented in Fig. 2a is divided into five phases [39, 40], each with several methods. Specifically, (1) image pre-processing is a standardized operation before using image data. It mainly includes 6 methods, such as histogram equalization [41], image enhancement [42], and image registration [43]. (2) Segmentation is the extraction of regions of interest in images, which can be divided into automatic, semi-automatic, and manual segmentation methods, such as edge-based segmentation [44], K-means clustering [45], and fuzzy C-means clustering [46]. (3) Feature extraction is performed on the original image (OI) or nine derived images (DI) processed by the filter. There are four types of features: first-order statistical features (FSF) [47], shape-based features [48], texture-based features (TF) [49], and wavelet features (WF) [50]. Note that the FSF and WF are extracted on the OI and nine DIs. WF is calculated on 8 sub-bands of OI for FSF and TF. Thus, there are 10 + 1 + 10 + 16 = 37 methods. (4) Feature selection filters out redundant and unstable features. It mainly contains 8 methods, such as LASSO regression [51] and minimum redundancy maximum relevance [52]. (5) Model construction is the selection of a suitable model based on various target problems, which mainly includes 4 methods, such as logistic regression [53] and support vector machine (SVM) [54]. In Ref. [38], the DGG model first segments the three sequences and extracts features in ROI by four methods. The features are then filtered by the LASSO algorithm and modeled using SVM. Thus, the subtask set after modeling the DGG model as the corresponding DAG is $M = \{m_{d_1,1}, m_{d_1,2}, m_{d_1,3}, m_{d_1,4}, m_{d_1,5}, m_{d_1,6}, m_{d_1,7}, m_{d_1,8}\}$ (Fig. 2b). Directed lines indicate data dependencies between its subtasks. For example, $m_{d_1,7}$ must run after the processing of $m_{d_1,3}$, $m_{d_1,4}$, $m_{d_1,5}$ and $m_{d_1,6}$.

Local computing

As shown in Fig. 3a, we construct the DAG topology for the RIDM task. In the local computing mode (LCM), the $m_{d_i,v}$ in g_{d_i} is only performed locally on the terminal device, with the offloading proportion $\mu_{d_i,v} = 0$. To clearly explain the parameters of LCM under g_{d_i} topology, it is formalized as follows:

Definition 3 Local computing parameters model is 4-tuple: $V^l = (F^l, \chi^l, T^l, E^{l,c})$, which is described in Appendix. For terminal device d_i, the local computation delay $\tau^{l,c}_{d_i,v}$ [s] of processing $b^l_{d_i,v}$ can be given by

$$\tau^{l,c}_{d_i,v} = \left(b^l_{d_i,v} \cdot \phi\right) / f^l_{d_i} \tag{1}$$

The local actual execution start time $\mathrm{st}^{l,c}_{d_i,v}$ [s] of processing $b^l_{d_i,v}$ on d_i can be given by

$$\mathrm{st}^{l,c}_{d_i,v} = \max\left\{\mathrm{it}^{l,c}_{d_i,v}, \mathrm{psc}^{l,c}_{d_i,v}\right\} \tag{2}$$

where $\mathrm{it}^{l,c}_{d_i,v} = \max\left\{\mathrm{it}^{l,c}_{d_i,v-1}, \mathrm{ft}^{l,c}_{d_i,v-1}\right\}$ denotes the idle time of the CPU d_i when executing $m_{d_i,v}$. $\mathrm{psc}^{l,c}_{d_i,v} = \max_{g \in \mathrm{pred}(v)}\left\{\mathrm{ft}^{s,d}_{e_{j,d_i,g}}, \mathrm{ft}^{l,c}_{d_i,g}\right\}$ indicates the last predecessor subtask of $m_{d_i,v}$ has been completed. pred (v) is the set of predecessor subtasks of $m_{d_i,v}$. $\mathrm{ft}^{s,d}_{e_{j,d_i,g}}$ see "ECCM workflow". Therefore, the outer max block in (2) represents that $m_{d_i,v}$ starts execution on d_i if and only if pred (v) has completed and CPU d_i is idle. Hence, the local actual execution finish time $\mathrm{ft}^{l,c}_{d_i,v}$ [s] of processing $b^l_{d_i,v}$ on d_i can be given by

$$\mathrm{ft}^{l,c}_{d_i,v} = \mathrm{st}^{l,c}_{d_i,v} + \tau^{l,c}_{d_i,v} \tag{3}$$

Table 1 System notations description

Notation	Description	Notation	Description	Notation	Description
e_j	j-th edge server	d_i	Device used by the i-th user	g_{d_i}	DAG topology of RIDM required for d_i
$b_{d_i,v}$	Subtask data size of $m_{d_i,v}$	$b^l_{d_i,v}$	Data size for running locally	$b^u_{d_i,v}$	Data size of subtask sent to the edge server
$b^c_{d_i,v}$	Data size for computing on the edge server	$b^d_{d_i,v}$	Data size of the result received	μ_{d_i}	Finite set of subtask offloading strategies for a g_{d_i} of d_i
V^l	Local computing parameters	ϕ	Task computational complexity	ζ	Mapping from TD to ES
$\mu_{d_i,v}$	$\mu_{d_i,v} \in \{0, 1\}$, The offloading strategy for the v-th subtask of d_i	$\tau^{total}_{d_i}$	Total delay required by d_i to process all $b_{d_i,v}$ of g_{d_i} given an offloading strategy μ_{d_i}	$\psi^{total}_{d_i}$	Total energy consumption required by d_i to process all $b_{d_i,v}$ of a g_{d_i} given a μ_{d_i}
V^s	Edge collaboration computing parameters	$m_{d_i,v}$	v-th medical subtask of d_i in g_{d_i}	$\vec{z}$	Dependency between $m_{d_i,v}$ and $m_{d_i,w}$
$f^l_{d_i}$	Computational capability of d_i	$\chi^l_{d_i}$	Consumed energy per CPU cycle of d_i	$\psi^{l,c}_{d_i,v}$	Computation energy consumption required by d_i to process $b^l_{d_i,v}$ locally
$\tau^{l,c}_{d_i,v}$	Computation delay required by d_i to process $b^l_{d_i,v}$ locally	$\mathrm{st}^{l,c}_{d_i,v}$	Actual local execution start time for $b^l_{d_i,v}$ processing on d_i	$\mathrm{ft}^{l,c}_{d_i,v}$	Actual local execution finish time for $b^l_{d_i,v}$ processing on d_i
$f^s_{e_j}$	Computational capability of e_j	$\chi^s_{e_j}$	Consumed energy per CPU cycle of e_j	$\Delta m^s_{e_j}$	Residual energy of e_j
$\Delta f^s_{e_j}$	Residual computational capabilities of e_j	$p^s_{e_j,d_i}$	Transmission power between d_i and e_j	$r^s_{e_j,d_i}$	Transmission rate between d_i and e_j
$\tau^{s,u}_{e_j,d_i,v}$	Transmission delay required by d_i to send $b^u_{d_i,v}$ to e_j via the uplink channel	$\mathrm{st}^{s,u}_{e_j,d_i,v}$	Actual execution start time on the uplink channel when sending $b^u_{d_i,v}$	$\mathrm{ft}^{s,u}_{e_j,d_i,v}$	Actual execution finish time on the uplink channel when sending $b^u_{d_i,v}$
$\tau^{s,c}_{e_j,d_i,v}$	Computation delay required by d_i to process $b^c_{d_i,v}$ on e_j	$\mathrm{st}^{s,c}_{e_j,d_i,v}$	Actual execution start time for $b^c_{d_i,v}$ processing on e_j	$\mathrm{ft}^{s,c}_{e_j,d_i,v}$	Actual execution finish time for $b^c_{d_i,v}$ processing on e_j
$\tau^{s,d}_{e_j,d_i,v}$	Transmission delay required by d_i to receive $b^d_{d_i,v}$ on e_j via the uplink channel	$\mathrm{st}^{s,d}_{e_j,d_i,v}$	Actual execution start time to e_j via the downlink channel when receiving the result data $b^d_{d_i,v}$	$\mathrm{ft}^{s,d}_{e_j,d_i,v}$	Actual execution finish time to e_j via the downlink channel when receiving the result data $b^d_{d_i,v}$
$\psi^{s,u}_{e_j,d_i,v}$	Transmission energy consumption to e_j via the uplink channel when sending $b^u_{d_i,v}$	$\psi^{s,c}_{e_j,d_i,v}$	Computation energy consumption required by d_i to process $b^c_{d_i,v}$ on e_j	$\psi^{s,d}_{e_j,d_i,v}$	Transmission energy consumption to e_j via the downlink channel when receiving $b^d_{d_i,v}$

Besides the required computation delay, processing each subtask also generates some computation energy. The local computation energy consumption $\psi^{l,c}_{d_i,v}$ [J] required by d_i to process $b^l_{d_i,v}$ can be given by

$$\psi^{l,c}_{d_i,v} = \chi^L_{d_i} \cdot b^l_{d_i,v} \cdot \phi \tag{4}$$

Figure 3b illustrates the execution process of the subtask locally at d_i. The factors that determine the local actual execution finish time of $m_{d_i,v}$ are the local actual start time and the local computation delay. In addition, the local computation energy consumption during execution is also essential.

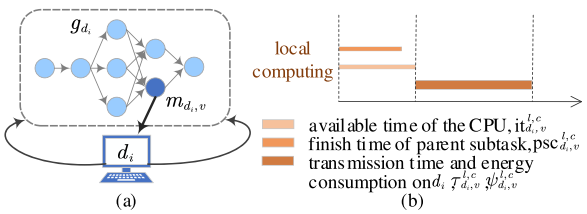


Fig. 3 Local computing mode. The width and length of the rectangle represent the energy consumption and delay generated during this stage

Edge collaboration computing

Figure 4b demonstrates the edge collaborative computing mode (ECCM) network architecture. The architecture comprises multiple heterogeneous ESs. Each ES has equal rights to share computing and communication resources at the edge of the network. Formally, we model the network architecture as an undirected graph, $G^s = \{N^s, C^s\}$, where the vertex set $N^s = E$ is the ESs in the network and the edge set C^s denotes the connections among ESs, respectively. $c_{jk} = (e_j, e_k) \in C^s$ represents the connection between e_j

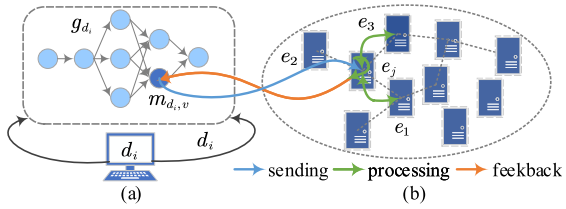


Fig. 4 Edge collaboration computing mode

and e_k. Assuming that d_i falls within the communication range of e_j, as shown in Fig. 4. $m_{d_i,v}$ in g_{d_i} is offloaded and runs in governed e_j under the ζ function mapping, where offloading proportion $\mu_{d_i,v} = 1$. To clearly explain the parameters of ECCM with g_{d_i} topology, it is formalized as follows:

Definition 4 Edge collaboration computing parameters model is 10-tuple: $V^s = (F^s, \chi^s, \Delta M^s, \Delta F^s, P^s, R^s, T^u, T^c, T^d, E^s)$, which is described in Appendix.

ECCM is a three-step process that includes sending, processing, and feedback. $m_{d_i,v}$ is first sent from d_i to e_j. Second, to enhance processing speed, e_j adopts a DCP algorithm to find suitable adjacent ESs at the edge layer. Subsequently, e_j executes $m_{d_i,v}$ in a distributed manner with these adjacent ESs. Finally, the result of the processing is feedback to d_i. The detailed workflow of ECCM is described in "ECCM workflow".

Problem formulation

The goal of the three-layer hierarchical system is to find an effective offloading strategy to maximize the utility of g_{d_i} after RIDM task execution. The total delay and total energy consumption are affected by the resources of d_i and e_j and the execution location of subtasks. The total delay $\tau_{d_i}^{total}$ [s] required to process all data of a g_{d_i} can be given by

$$\tau_{d_i}^{total} = \max\left[\max_{q\in \mathrm{EMT}}\left(\mathrm{ft}_{e_j,d_i,q}^{s,d}, \mathrm{ft}_{d_i,q}^{l,c}\right)\right] \tag{5}$$

where EMT is the set of exit medical subtasks that are without successor subtasks. The total energy consumption $\psi_{d_i}^{total}$ [J] required to process all data of a g_{d_i} can be given by

$$\psi_{d_i}^{total} = \sum_{v=1}^{V}\left(\psi_{d_i,v}^{l,c}\cdot\left(1-\mu_{d_i,v}\right), \left(\psi_{e_j,d_i,v}^{s,u} + \psi_{e_j,d_i,v}^{s,c} + e_{e_j,d_i,v}^{s,d}\right)\cdot\mu_{d_i,v}\right) \tag{6}$$

where $\mathrm{ft}_{e_j,d_i,q}^{s,d}$, $\psi_{e_j,d_i,v}^{s,u}$, $\psi_{e_j,d_i,v}^{s,c}$, $e_{e_j,d_i,v}^{s,d}$ see "ECCM workflow". The weighted sum of delay and energy consumption, i.e., utility, is used as the performance metric in this article. Let β^t and β^e be the weight indicators, where $\beta^t + \beta^e = 1$ and $\beta^t, \beta^e \in [0, 1]$. For the terminal device d_i, the utility of a g_{d_i} given an offloading strategy μ_{d_i}, $O_{\mu_{d_i}}^C$ is given by

$$O_{\mu_{d_i}}^C = \beta^t\cdot\frac{\max_{q\in\mathrm{EMT}}\mathrm{ft}_{d_i,q}^{l,c} - \tau_{d_i}^{total}}{\max_{q\in\mathrm{EMT}}\mathrm{ft}_{d_i,q}^{l,c}} + \beta^e\cdot\frac{\sum_{v=1}^{V}\psi_{d_i,v}^{l,c} - \psi_{d_i}^{total}}{\sum_{v=1}^{V}\psi_{d_i,v}^{l,c}} \tag{7}$$

where $\max_{q\in\mathrm{EMT}}\mathrm{ft}_{d_i,q}^{l,c}$ and $\sum_{v=1}^{V}\psi_{d_i,v}^{l,c}$ are the total delay and total energy consumption of the local execution of the RIDM task. Hence, the optimization problem with respect to the utility is formulated as follows:

$$\max\ O_{\mu_{d_i}}^C \tag{8}$$

Intuitively, the optimization problem in (8) is an NP-hard problem [55]. Finding the optimal offloading strategy can be extremely challenging for high-dynamic DAG topology. To tackle the challenges, this article proposes the DCDO-DRL strategy in "DCDO-DRL design".

ECCM workflow

This subsection describes the three stages of ECCM, including uploading subtasks, running subtasks on edge servers, and receiving the result data from subtasks.

Uploading subtasks

In the ECCM, the subtask is first uploaded from TD to ES and executed then on the edge server instead of locally. The transmission delay $\tau_{e_j,d_i,v}^{s,u}$ [s] required by d_i to send $b_{d_i,v}^u$ to e_j via the uplink channel can be given by

$$\tau_{e_j,d_i,v}^{s,u} = b_{d_i,v}^u / r_{e_j,d_i}^s \tag{9}$$

The actual execution start time $\mathrm{st}_{e_j,d_i,v}^{s,u}$ [s] of sending $b_{d_i,v}^u$ on the uplink channel can be given by

$$\mathrm{st}_{e_j,d_i,v}^{s,u} = \max\left\{\mathrm{it}_{e_j,d_i,v}^{s,u}, \mathrm{psc}_{e_j,d_i,v}^{s,u}\right\} \tag{10}$$

where $\mathrm{it}_{e_j,d_i,v}^{s,u} = \max\left\{\mathrm{it}_{e_j,d_i,v-1}^{s,u}, \mathrm{ft}_{e_j,d_i,v-1}^{s,u}\right\}$ is the idle time on the uplink channel when sending $m_{d_i,v}$. $\mathrm{psc}_{e_j,d_i,v}^{s,u} = \max_{g\in\mathrm{pred}(v)}\left\{\mathrm{ft}_{e_j,d_i,g}^{s,d}, \mathrm{ft}_{d_i,g}^{l,c}\right\}$ represents all data needed by $m_{d_i,v}$ has finished. Therefore, the outer max block in (10) denotes that $m_{d_i,v}$ is allowed to send data to e_j if and only if the idle time of the uplink channel and pred (v) has completed.

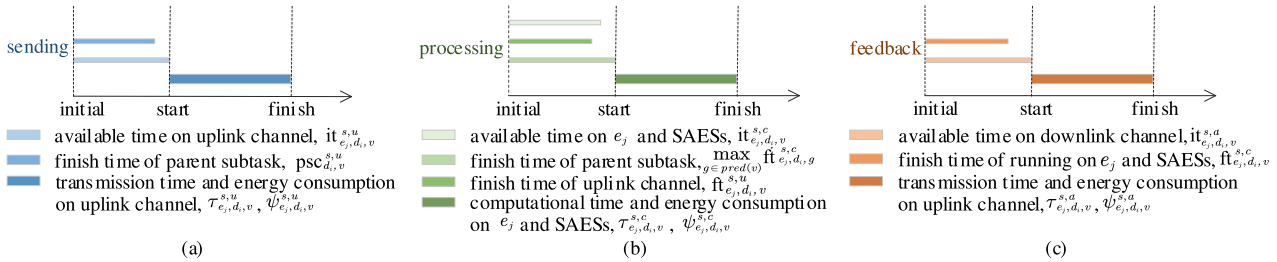


Fig. 5 The execution process of subtasks on the ECCM. The width and length of the rectangle represent the energy consumption and delay generated during this stage

The actual execution finish time $ft^{s,u}_{e_j,d_i,v}$ [s] of sending $b^u_{d_i,v}$ on the uplink channel can be given by

$$ft^{s,u}_{e_j,d_i,v} = st^{s,u}_{e_j,d_i,v} + \tau^{s,u}_{e_j,d_i,v} \tag{11}$$

The transmission energy consumption $\psi^{s,u}_{e_j,d_i,v}$ [J] on the uplink channel when sending $b^u_{d_i,v}$ can be given by

$$\psi^{s,u}_{e_j,d_i,v} = p^s_{e_j,d_i} \cdot \tau^{s,u}_{e_j,d_i,v} = \left(p^s_{e_j,d_i} \cdot b^u_{d_i,v}\right) / r^s_{e_j,d_i} \tag{12}$$

Figure 5a shows the sending phase in the ECCM, illustrating the process of $m_{d_i,v}$ is uploaded from d_i to affiliated e_j. At this phase, the factors that determine the actual execution finish time of $m_{d_i,v}$ are the actual start time and transmission delay in the upload channel. In addition, there is the transmission energy consumption.

Running on edge servers

Inspired by Ref. [8], we propose a DCP algorithm at the edge layer to accelerate subtask processing by exploring the collaborative computing capabilities between ESs. DCP algorithm avoids the problem that a long processing time on a single ES with limited computational resources.

To describe the DCP algorithm more clearly, the pseudocode and diagram are shown in Algorithm 1 and Fig. 6. The DCP algorithm includes three parts: the first part is lines 1–6, located in the adjacent ESs for e_j. The adjacent ESs are defined based on whether there is an edge between two ESs in the edge layer network topology $G^s = \{N^s, C^s\}$. The second part is lines 7–11, finding the suitable adjacent ESs (SAESs). Filtering adjacent ES by determining if there is enough remaining memory in the ES to execute subtasks. The third part is lines 12–17, the subtask is further divided into small subtasks according to the subtask allocation matrix U and the remaining computing capacity of ESs. SAESs cooperatively process the assigned small subtask, calculate computational delay and computational energy consumption, and return results to e_j.

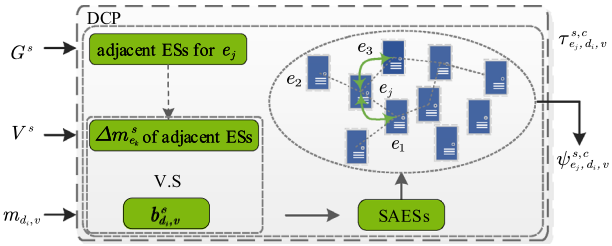


Fig. 6 The execution process of subtasks on the ECCM

The total computational power available F for subtask execution at the edge layer is

$$F = f^s_{e_j} + \sum_{u_{jk=1}} \Delta f^s_{e_k} \quad \forall k = 1, 2, ...m \tag{13}$$

Note that fiber optic communication with a high transmission rate is used between edge servers. Thus, the transmission delay is negligible when the adjacent ESs receive assigned small subtasks and send processing results back to e_j.

The computation delay $\tau^{s,c}_{e_j,d_i,v}$ [s] required by d_i to process $b^c_{d_i,v}$ on the SAESs can be given by

$$\tau^{s,c}_{e_j,d_i,v} = \left(b^s_{d_i,v} \cdot \phi\right) / F \tag{14}$$

Similarly, the actual execution start time $st^{s,c}_{e_j,d_i,v}$ [s] for $b^c_{d_i,v}$ processing on the SAESs can be given by

$$st^{s,c}_{e_j,d_i,v} = \max\left\{it^{s,c}_{e_j,d_i,v}, psc^{s,c}_{e_j,d_i,v}\right\} \tag{15}$$

where $pcs^{s,c}_{e_j,d_i,v} = \max\left\{\max_{g\in pred(v)} ft^{s,c}_{e_j,d_i,g}, ft^{s,u}_{e_j,d_i,v}\right\}$ indicates that $b^c_{d_i,v}$ has been uploaded to e_j and all the predecessor data needed by $m_{d_i,v}$ has finished. $it^{s,c}_{e_j,d_i,v} = \max\left\{it^{s,c}_{e_j,d_i,v-1}, ft^{s,c}_{e_j,d_i,v-1}\right\}$ is the idle time that e_j can handle $b^c_{d_i,v}$. Notice that since we assign small subtasks to ES in SAESs based on computational capacity, the execution time of each small subtask is guaranteed to be the same. Thus, it ensures the consistency of the idle time of e_j and ES in SAESs. Therefore, the outer max block in (15) indicates that

Algorithm 1 DCP

Require: the size of offloaded, $b^c_{d_i,v}$; ESs network topology, $G^s = \{N^s, C^s\}$; Properties of the ESs, V^s

$U \leftarrow 0$ ▷ Initialize subtask allocation matrix
for each $e_k \in N^e$ **do**
  if c_{jk} exist **then**
    ▷ Judge communication among ESs to find the adjacent ESs of e_j
    $u_{jk} = 1$ ▷ Update subtask allocation matrix
  end if
end for
for each $u_{jk} = 1$ **do**
  if $b^c_{d_i,v} > \Delta m^s_{e_k}$ exist **then** ▷ Judge remaining memory of e_k to find SAESs
    $u_{jk} = 0$ ▷ Update subtask allocation matrix
  end if
end for
▷ SAESs and e_j run subtask in a distributed manner
for each $u_{jk} = 1$ **do**
  Compute total computing power by (13)
  Compute computation energy consumption by (17)
end for
Compute the computation delay by (14)

Ensure: computation delay and computation energy consumption $\tau^{s,c}_{e_j,d_i,v}$, $\psi^{s,c}_{e_j,d_i,v}$

the actual start time that $m_{d_i,v}$ only can be processed relies on the idle time of e_j and the actual execution finish time of predecessor subtasks.

The actual execution finish time $\mathrm{ft}^{s,c}_{e_j,d_i,v}$ [s] for $b^c_{d_i,v}$ processing on the SAESs can be given by

$$\mathrm{ft}^{s,c}_{e_j,d_i,v} = \mathrm{st}^{s,c}_{e_j,d_i,v} + \tau^{s,c}_{e_j,d_i,v} \tag{16}$$

Meanwhile, computation energy consumption is also generated. The computation energy consumption $\psi^{s,c}_{e_j,d_i,v}$ [J] required by d_i to process $b^c_{d_i,v}$ on the SAESs can be given by

$$\psi^{s,c}_{e_j,d_i,v} = \left(\frac{\chi^s_{e_j} \cdot f^s_{e_j}}{F} + \sum_{u_{jk}=1} \frac{\chi^s_{e_k} \cdot f^s_{e_k}}{F} \right) \cdot b^s_{d_i,v} \cdot \phi \quad \forall k = 1, 2, \ldots m \tag{17}$$

The subtasks are executed in a distributed manner by ES and the SAESs, as shown in Fig. 6. Figure 5b illustrates, during the processing phase, the actual execution finish time of $m_{d_i,v}$ lies on the actual start execution time and computation time on e_j. In addition, there is the computational energy consumption.

Receiving the result data

After the subtask is executed on ES, the results data will be sent back to TD. The transmission delay $\tau^{s,d}_{e_j,d_i,v}$ [s] required to receive the result data $b^d_{d_i,v}$ from e_j to d_i via the downlink channel can be given by

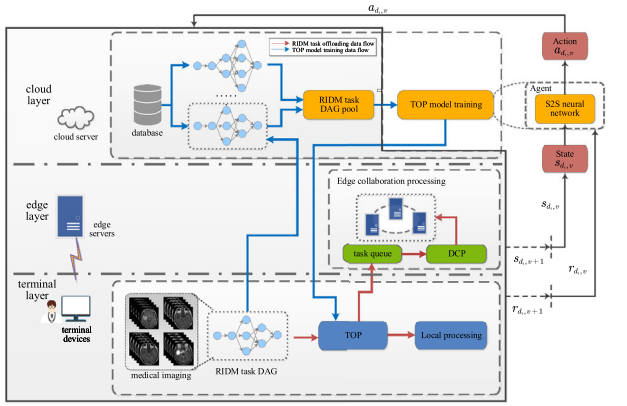


Fig. 7 The framework of DCDO-DRL. (1) TOP model training data flow. The TOP model uses an S2S neural network to interact with the environment and optimize the offloading strategy through RL. (2) RIDM task offloading data flow. TD loads the trained TOP model from the cloud to obtain subtask offloading locations and execute subtasks accordingly

$$\tau^{s,d}_{e_j,d_i,v} = b^d_{d_i,v} / r^s_{e_j,d_i} \tag{18}$$

Similarly, the actual execution start time $\mathrm{st}^{s,d}_{e_j,d_i,v}$ [s] of receiving $b^d_{d_i,v}$ on the downlink channel can be given by

$$\mathrm{st}^{s,d}_{e_j,d_i,v} = \max \left\{ \mathrm{ft}^{s,c}_{e_j,d_i,v}, \mathrm{it}^{s,d}_{e_j,d_i,v} \right\} \tag{19}$$

where $\mathrm{it}^{s,d}_{e_j,d_i,v} = \max \left\{ \mathrm{it}^{s,d}_{e_j,d_i,v-1}, \mathrm{ft}^{s,d}_{e_j,d_i,v-1} \right\}$ is the idle time on the downlink channel when receiving the result. Therefore, the outer max block in (19) denoted that the actual start time that $m_{d_i,v}$ only can return result data to d_i depends on the idle time of the downlink channel and the actual execution finish time of $m_{d_i,v}$ in the processing phase.

The local actual execution finish time $\mathrm{ft}^{s,d}_{e_j,d_i,v}$ [s] of receiving $b^d_{d_i,v}$ on the downlink channel can be given by

$$\mathrm{ft}^{s,d}_{e_j,d_i,v} = \mathrm{st}^{s,d}_{e_j,d_i,v} + \tau^{s,d}_{e_j,d_i,v} \tag{20}$$

The transmission energy consumption $\psi^{s,d}_{e_j,d_i,v}$ [J] required to receive the result data $b^d_{d_i,v}$ from ES to d_i via the downlink channel can be given by

$$\psi^{s,d}_{e_j,d_i,v} = p^s_{e_j,d_i} \cdot \tau^{s,d}_{e_j,d_i,v} = \left(p^s_{e_j,d_i} \cdot b^d_{d_i,v} \right) / r^s_{e_j,d_i} \tag{21}$$

The execution of the results data is sent from the governed ES to the terminal device as shown in Fig. 5c. Similarly, during the feedback phase, the actual execution finish time of $m_{d_i,v}$ depends on the actual execution start time and transmission time on the download channel. In addition, there is the transmission of energy consumption.

DCDO-DRL design

This section describes first the architecture of the DCDO-DRL strategy. Next, the RIDM task offloading problem is formulated as a Markov decision process (MDP). Then, A S2S neural network is adopted to predict the offloading process. Finally, we introduce the workflow mechanism of the DCDO-DRL strategy.

DCDO-DRL

According to the challenges introduced in "Introduction", we optimized the system model in "System model and problem formulation" and constructed the DCDP-DRL strategy, whose architecture is shown in Fig. 7. Each TD is equipped with a TOP module derived from the cloud-trained model. The edge collaborative processing module executes subtasks assigned in a distributed manner. There are two components in the could layer: (1) RIDM task DAG pool stores DAGs from the different RIDM tasks of TDs. (2) TOP model training module outputs the offload location of the subtask via RL and S2S neural networks.

The DCDO-DRL architecture includes two data flows. (1) TOP model training data flow. The TD first embeds the data information of the RIDM task into DAG; Next, the DAG is uploaded to the RIDM task DAG pool; Finally, the S2S neural network (agent) in the TOP model interacts with the environment (network, DAG, as well as the computing power of TDs and ES) to iteratively learn and optimize the offloading strategy. (2) RIDM task offloading data flow. The TD first loads the TOP model trained in the cloud; then, the test DAG on the TD is input to the TOP model to get the execution location of subtasks, i.e., local processing or edge distributed processing.

MDP formulation

To deal with the RIDM task offloading problem, we adopt a DRL-based algorithm to get an offloading strategy to maximize the utility of the RIDM task execution. First, the offloading problem is formulated as an MDP to implement the DRL algorithm. In this article, the MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma)$, where $\mathcal{S}$ is the environment states space, $\mathcal{A}$ is the action space, $\mathcal{R}$ is reward function, $\mathcal{P}$ is the state transition probability matrix and γ is the discount factor. The motivation of an agent is to find a strategy that can maximize accumulated reward and select the best behavior. Hence, the three key elements of MDP can be defined as follows:

State: The DEC cost of performing $m_{d_i,v}$ is related to the RIDM topologies g_{d_i}, the task size B, the task computational complexity ϕ, local computing mode parameters V^l, edge collaboration computing mode parameters V^s, etc.. Thus, the state space reflects the observations from the environment when the RIDM task executes, which can be given by

$$\mathcal{S} = \left\{ s_{d_i,v} | i = 1, 2, \ldots, n; v = 1, 2, \ldots, V \right\} \tag{22}$$

where $s_{d_i,v} = \left(Em\left(g_{d_i}\right), \mu_{d_i,1:v} \right)$ denotes the state when running $m_{d_i,v}$. $\mu_{d_i,1:v} = \left\{ \mu_{d_i,1}, \mu_{d_i,2}, ..., \mu_{d_i,v} \right\}$ is the partial offloading decision for the subtasks from $m_{d_i,1}$ to $m_{d_i,v}$. $Em\left(g_{d_i}\right)$ is the encoded g_{d_i} with a sequence of subtask embedding. Each subtask embedding is a three-vector. The first vector is the indices of the immediate predecessor of $m_{d_i,v}$; the second vector contains the index of $m_{d_i,v}$ and DEC cost of $m_{d_i,v}$; the last vector is the indices of the immediate successor of $m_{d_i,v}$.

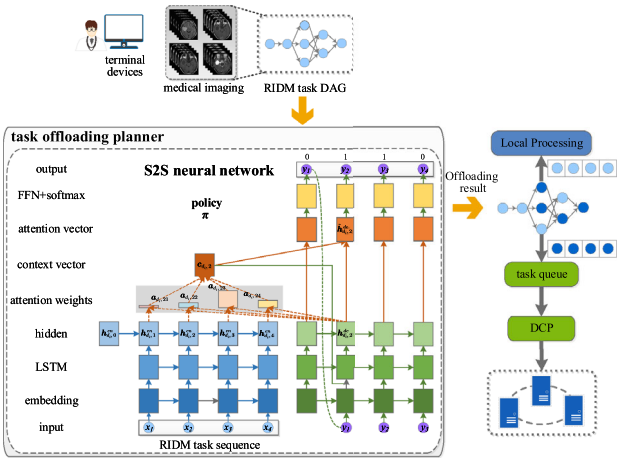


Fig. 8 Subtask offloading process. TD inputs the subtask sequence for DAG to the encoder in the S2S neural network with an attention mechanism. The decoder then outputs offloading locations based on this input, which are used to execute the subtasks accordingly

Action: Based on the observed environment states, the agent has two executions for each subtask, i.e., local execution or offloading to the edge server, so the action space can be given by $\mathcal{A} = \{0, 1\}$, $a_{d_i,v} = \mu_{d_i,v} = 0$ denotes local processing and $a_{d_i,v} = \mu_{d_i,v} = 1$ denotes edge collaboration processing.

Reward: According to the environment state and action, the agent calculates reward values. The objective is to maximize utility by (8). Utility is the weighted sum of DECs generated after the completion of multiple RIDM subtasks. The reward should guide the agreement between the objective and learning. To achieve this objective, we define the reward function as an increment of the DEC after making an offloading decision for a subtask. There are four reasons. First, we have to consider the DEC to ensure maximum utility without sacrificing one factor. Second, the weight can flexibly adjust the proportion of DEC. Third, the function helps to prevent the agent from getting stuck and adapting to changes in the envi-

ronment. Finally, the increment measures the consequences of offloading decisions, facilitating a balance between global and local utility. Formally, the reward function can be given by

$$r_{d_i,v} = \beta^t \cdot \frac{\left(\max_{q\in \mathrm{EMT}} \mathrm{ft}^{l,c}_{d_i,q}\right)/V - \left(\tau^{total}_{d_i,1:v} - \tau^{total}_{d_i,1:v-1}\right)}{\max_{q\in \mathrm{EMT}} \mathrm{ft}^{l,c}_{d_i,q}} + \beta^e \cdot \frac{\left(\sum_{v=1}^{V} \psi^{l,c}_{d_i,v}\right)/V - \left(\psi^{total}_{d_i,1:v} - \psi^{total}_{d_i,1:v-1}\right)}{\sum_{v=1}^{V} \psi^{l,c}_{d_i,v}} \tag{23}$$

where $\max_{q\in \mathrm{EMT}} \mathrm{ft}^{l,c}_{d_i,q}$ and $\sum_{v=1}^{V} \psi^{l,c}_{d_i,v}$ are the delay and energy consumption required to run all tasks in the DAG locally. $\tau^{total}_{d_i,1:v} - \tau^{total}_{d_i,1:v-1}$ and $\psi^{total}_{d_i,1:v} - \psi^{total}_{d_i,1:v-1}$ are the increment of the delay and energy consumption.

Subtask offloading process

According to (8) and MDP, the sequential decision-making of the RIDM task offloading problem is switched to an S2S prediction problem. The input of the S2S neural network is a sequence of subtask embedding. the output is an offloading strategy $\pi\left(\mu_{d_i} | Em\left(g_{d_i}\right)\right)$. The strategy is the probability of V subtasks selecting action given the encoded g_{d_i}, which can be given by

$$\begin{aligned} &\pi\left(\mu_{d_i} | Em\left(g_{d_i}\right)\right) \\ &= \prod_{v=1}^{V} \pi\left(\mu_{d_i,v} | Em\left(g_{d_i}\right), \mu_{d_i,v-1}\right) \\ &= \prod_{v=1}^{V} \mathbb{P}\left(\mu_{d_i,v} | Em\left(g_{d_i}\right), \mu_{d_i,v-1}\right) \end{aligned} \tag{24}$$

where $\mathbb{P}\left(\mu_{d_i,v} | Em\left(g_{d_i}\right), \mu_{d_i,v-1}\right)$ is the probability of selecting action $\mu_{d_i,v}$ for $m_{d_i,v}$ under the state $s_{d_i,v}$. The subtask offloading process includes three steps, as shown in Fig. 8.

Step 1: Get the subtask sequence for g_{d_i}. We arrange all the subtasks by (25). The central idea is to choose the maximum weight-sum of running delay and energy consumption for each subtask under LCM and ECCM. The indexes of all subtasks are then sorted in ascending order by sort value, where $succ(v)$ is the set of successor subtasks of $m_{d_i,v}$. $O^l_{d_i,v} = \tau^{l,c}_{d_i,v} + \psi^{l,c}_{d_i,v}$ is the running delay and energy consumption locally. $\tau^s_{e_j,d_i,v} = \tau^{s,u}_{e_j,d_i,v} + \tau^{s,c}_{e_j,d_i,v} + \tau^{s,d}_{e_j,d_i,v}$ and $\psi^s_{e_j,d_i,v} = \psi^{s,u}_{e_j,d_i,v} + \psi^{s,c}_{e_j,d_i,v} + e^{s,d}_{e_j,d_i,v}$ indicate the running delay and energy consumption of $m_{d_i,v}$ during the upload, processing, and feedback phases of ECCM.

$$\mathrm{sort}\left(m_{d_i,v}\right) = \begin{cases} \min\left(O^l_{d_i,v}, \tau^s_{e_j,d_i,v} + \psi^s_{e_j,d_i,v}\right) & , \text{if } v \in \mathrm{EMT} \\ \min\left(O^l_{d_i,v}, \tau^s_{e_j,d_i,v} + \psi^s_{e_j,d_i,v} + \max_{q\in succ(v)}\left(\mathrm{sort}\left(m_{d_i,q}\right)\right)\right) & , \text{if } v \notin \mathrm{EMT} \end{cases} \tag{25}$$

Step 2: Input the subtask sequence to the encoder of the S2S neural network. Once the encoding is done, feed it to the decoder and get the output.

The offloading strategy defined in (25) can be represented by an S2S neural network. In this article, we adopt Bidirectional Long Short-Term Memory (Bi-LSTM) [56] and Long Short-Term Memory (LSTM) [57] as an encoder and a decoder in a S2S neural network. The encoder of the S2S neural network converts the input graph into a continuous subtask sequence $M = \left\{m_{d_i,v} |\ v = 1, 2, \ldots, V\right\}$. A decoder then uses this sequence to generate the offloading strategy $\mu_{d_i} = \left\{\mu_{d_i,v} | v = 1, 2, \ \ldots, V\right\}$. This combination can integrate node features and relationships, capture global context and handle long-term dependencies effectively. The details are as follows: $m_{d_i,v}$ is first converted to an embedding vector $\boldsymbol{m}_{d_i,v}$ before each encoding step, and then the Bi-LSTM transforms the hidden state $\boldsymbol{h}^{en}_{d_i,v-1}$ at the previous step and $\boldsymbol{m}_{d_i,v}$ into the hidden state $\boldsymbol{h}^{en}_{d_i,v}$ at the current step encoder, which can be given by

$$\boldsymbol{h}^{en}_{d_i,v} = \text{Bi-LSTM}\left(\boldsymbol{h}^{en}_{d_i,v-1}, \boldsymbol{m}_{d_i,v}\right) \tag{26}$$

After the embedding vectors of all subtasks are encoded in sequence, the hidden layer state vector $\boldsymbol{h}^{en}_{d_i} = \left\{h^{en}_{d_i,v} |\ v = 1, 2, \ldots, V\right\}$ of an encoder is got.

To improve the efficiency and accuracy of task processing, we introduce an attention mechanism. The context vector $\boldsymbol{c}_{d_i,d}$ decoded in step d is the weighted average of all hidden states $\boldsymbol{h}^{en}_{d_i,v}$ of the encoder output, which can be given by

$$\boldsymbol{c}_{d_i,d} = \sum_{i=1}^{V} \frac{\exp\left(\mathrm{score}\left(\boldsymbol{h}^{de}_{d_i,d}, \boldsymbol{h}^{en}_{d_i,v}\right)\right)}{\sum_{k=1}^{V} \exp\left(\mathrm{score}\left(\boldsymbol{h}^{de}_{d_i,k}, \boldsymbol{h}^{en}_{d_i,v}\right)\right)} \cdot \boldsymbol{h}^{en}_{d_i,v} \tag{27}$$

where the weight $a_{d_i,d,v}$ is a probability distribution at $v = 1, 2, \ldots, V$ for a given d. score $\left(\boldsymbol{h}^{de}_{d_i,d}, \boldsymbol{h}^{en}_{d_i,v}\right)$ is a forward feedback neural network, which computes an alignment score from the hidden state $\boldsymbol{h}^{en}_{d_i,v}$ of encoder at step v and the hidden state $\boldsymbol{h}^{de}_{d_i,d}$ of decoder at step d. At each step of decoding, LSTM takes as inputs the hidden state $\boldsymbol{h}^{de}_{d_i,d-1}$ at the previous step and the context vector $\boldsymbol{c}_{d_i,d}$ at the current step d, the hidden state $\boldsymbol{h}^{de}_{d_i,d}$ of the decoder output can be given by

$$\boldsymbol{h}^{de}_{d_i,d} = \text{LSTM}\left(\boldsymbol{h}^{de}_{d_i,d-1}, \boldsymbol{c}_{d_i,d}\right) \tag{28}$$

Combine the current decoder hidden state $\boldsymbol{h}_{d_i,d}^{de}$ and context vector $\boldsymbol{c}_{d_i,d}$, we get the attention hidden state $\widetilde{\boldsymbol{h}}_{d_i,d}^{de}$, which can be given by

$$\widetilde{\boldsymbol{h}}_{d_i,d}^{de} = \tanh\left(W_c\left[\boldsymbol{c}_{d_i,d}; \boldsymbol{h}_{d_i,d}^{de}\right]\right) \tag{29}$$

The predictive distribution is produced from the attentional vector $\widetilde{\boldsymbol{h}}_{d_i,d}^{de}$ and softmax layer, which can be given by

$$p(\mu_{d_i,d}|\mu_{<d_i,d}, \mathrm{M}) = \mathrm{softmax}\left(W_s\widetilde{\boldsymbol{h}}_{d_i,d}^{de}\right) \tag{30}$$

Step 3: With the output of the decoder in the S2S neural network, i.e., the offloading decisions μ_{d_i} of a sequence that contains all the subtasks, each of which is placed on the corresponding device. If $\mu_{d_i,v} = 0, m_{d_i,v}$ is performed locally; if $\mu_{d_i,v} = 1, m_{d_i,v}$ is sent to the corresponding edge server e_j and processed in a distributed manner according to the DCP algorithm.

Training mechanism

The SAC algorithm proposed by Haarnoja et al. [58] maximizes the entropy while the expected reward. Inspired by the SAC, a training mechanism is designed for the DCDO-DRL strategy to learn a robust DRL algorithm. The mechanism follows the discrete SAC, which reconstructs the action space of SAC to adapt to task offloading scenarios. Next, we discuss the training mechanism. The objective function, compared to the traditional RL, considers the entropy item $\alpha\mathcal{H}\left(\pi\left(\cdot|s_{d_i,v}\right)\right)$ and concentrates on maximizing the accumulated reward. The definition is as follows:

$$\begin{aligned}
&\max_\pi \sum_{v=1}^{V} \mathbb{E}_{\left(s_{d_i,v},a_{d_i,v}\right)\sim\tau_\pi}\left[\left(r\left(s_{d_i,v}, a_{d_i,v}\right)\right.\right.\\
&\quad\left.\left.+\alpha\mathcal{H}\left(\pi\left(a_{d_i,v}|s_{d_i,v}\right)\right)\right)\gamma^{v-1}\right]\\
&= \max_\pi \sum_{v=1}^{V} \mathbb{E}_{\left(s_{d_i,v},a_{d_i,v}\right)\sim\tau_\pi}\left[\left(r\left(s_{d_i,v}, a_{d_i,v}\right)\right.\right.\\
&\quad\left.\left.-\alpha\log\left(\pi\left(a_{d_i,v}|s_{d_i,v}\right)\right)\right)\gamma^{v-1}\right]
\end{aligned} \tag{31}$$

where τ_π is the state-action trajectory distribution following the policy π; $\gamma \in [0, 1]$ is a discount factor used to distinguish the importance between current and future rewards; α is the temperature parameter that controls the stochastic of the optimal policy; $\mathcal{H}\left(\pi\left(\cdot|s_{d_i,v}\right)\right) = -\mathbb{E}_{\left(s_{d_i,v},a_{d_i,v}\right)\sim\tau_\pi}\log\left(\pi\left(a_{d_i,v}|s_{d_i,v}\right)\right)$ is the entropy of the policy distribution, which permits the exploration of additional solutions.

The optimal temperature α varies across tasks due to differences in reward. In addition, the policy is continuously updated during training, resulting in changes to the corresponding Q value and further affecting the choice of α. Therefore, to train the temperature α parameter dynamically, we will rewrite (31) with the mean entropy as a constraint and the transformed objective function as follows: [59]

$$\begin{aligned}
&\max_\pi \sum_{v=1}^{V} \mathbb{E}_{\left(s_{d_i,v},a_{d_i,v}\right)\sim\tau_\pi}\left[r\left(s_{d_i,v}, a_{d_i,v}\right)\gamma^{v-1}\right]\\
&\text{s.t. } \mathcal{H}\left(\pi\left(\cdot|s_{d_i,v}\right)\right) \geq \hat{\mathcal{H}} \ \ \forall v \in V
\end{aligned} \tag{32}$$

where $\hat{\mathcal{H}}$ is the minimum value of the average entropy over the sample. The objective of our policy is transformed to maximize the cumulative reward, provided the sample average entropy is no less than $\hat{\mathcal{H}}$. The optimal temperature α_v^* can be given by

$$\begin{aligned}
\alpha_v^* = \mathrm{argmin}_{\alpha_{d_i,v}} &\mathbb{E}_{a_{d_i,v}\sim\pi_{d_i,v}^*}\left[-\alpha_{d_i,v}\right.\\
&\left.\left(\log\left(\pi_{d_i,v}^*\left(a_{d_i,v}|s_{d_i,v}; \alpha_{d_i,v}\right)\right)\hat{\mathcal{H}}\right)\right]
\end{aligned} \tag{33}$$

$\pi_{d_i,v}^*\left(a_{d_i,v}|s_{d_i,v}; \alpha_{d_i,v}\right)$ denotes the temperature $\alpha_{d_i,v}$ when the action $a_{d_i,v}$ is chosen according to the optimal policy $\pi_{d_i,v}^*$ in state $s_{d_i,v}$. Thus, the temperature objective of solving the α_v^*, which can be given by

$$\begin{aligned}
\mathcal{L}(\alpha) = &\mathbb{E}_{a_{d_i,v}\sim\pi_{d_i,v}}\\
&\left[\alpha\left(\log\left(\pi_{d_i,v}\left(a_{d_i,v}|s_{d_i,v}\right)\right) - \hat{\mathcal{H}}\right)\right]
\end{aligned} \tag{34}$$

It can be observed that optimal policy and optimal strategies interact with each other and that both should be updated iteratively. Based on Ref. [60], the (32) is solved using a soft strategy iteration with policy evaluation and policy promotion. In the policy evaluation phase, the DCDO-DRL strategy constructs two functions by modifying Bellman backup: (1) the soft action-value function $Q_\pi(s, a)$ evaluates the Q-value given state-action pair under the policy π; (2) the soft state-value function $v_\pi(s)$ evaluates the value of a state under the policy π with the entropy term. The two functions can be given by

$$Q_\pi(s, a) = r(s, a) + \gamma\sum_{s'\in S}\mathcal{P}\left(s'|s, a\right)V_\pi\left(s'\right) \tag{35}$$

$$V_\pi(s) = \mathbb{E}_{a\sim\pi}\left[Q_\pi(s, a) - \alpha log(\pi(a|s))\right] \tag{36}$$

Then, the mean squared error is the soft Bellman error method. It updates the soft Q-network parameter ξ by measuring the Q-network and target Q-network, which can be given by

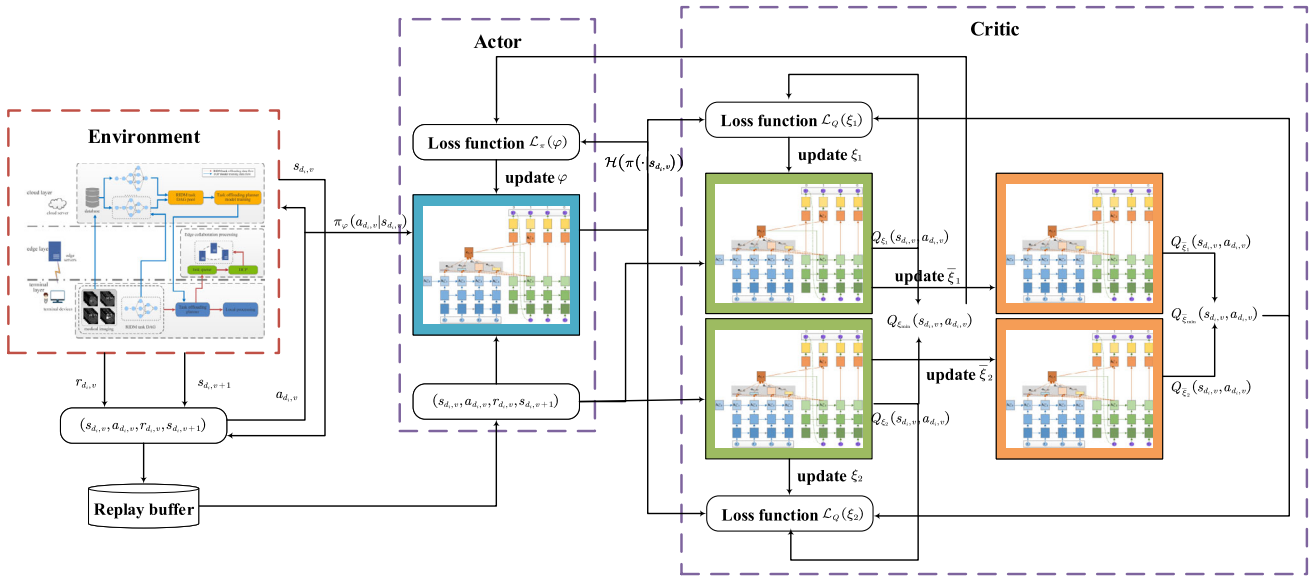


Fig. 9 The training mechanism of the DCDO-DRL strategy. (1) Actor network: maximizing expected returns. (2) Critical network: accurately estimating the value of each state-action pair. (3) Target network: its parameters gradually synchronize with the parameters of the main network during the training process to stabilize the learning process

$$\begin{aligned}\mathcal{L}_{\mathcal{Q}}(\xi) &= \mathbb{E}_{(s_{d_i,v},a_{d_i,v})\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\xi\left(s_{d_i,v},a_{d_i,v}\right)\right.\right.\\ &\quad\left.\left.-\acute{Q}\left(s_{d_i,v},a_{d_i,v}\right)\right)^2\right]\\ &= \mathbb{E}_{(s_{d_i,v},a_{d_i,v})\sim\mathcal{D}}\left[\frac{1}{2}\left(Q_\xi\left(s_{d_i,v},a_{d_i,v}\right)\right.\right.\\ &\quad\left.\left.-\left(r\left(s_{d_i,v},a_{d_i,v}\right)+\gamma V_{\bar{\xi}}\left(s_{d_i,v+1}\right)\right)\right)^2\right] \end{aligned} \tag{37}$$

where $\mathcal{D}$ is the replay buffer that stores a series of transitions $\left(s_{d_i,v}, a_{d_i,v}, r_{d_i,v}, s_{d_i,v+1}\right)\cdot\bar{\xi}$ is the parameter for a target Q-network and copied from ξ after a certain time.

Since the action space in this article is discrete, the expectation of $V_{\bar{\xi}}\left(s_{d_i,v+1}\right)$ can be solved by discrete action probabilities with random variable states, which can be given by

$$\begin{aligned}V_{\bar{\xi}}\left(s_{d_i,v+1}\right) &= \sum_{a_{d_i,v+1}\in\mathcal{A}}\pi\left(a_{d_i,v+1}|s_{d_i,v+1}\right)\\ &\left[Q_{\bar{\xi}}\left(s_{d_i,v+1},a_{d_i,v+1}\right)-\alpha\log\left(\pi\left(a_{d_i,v+1}|s_{d_i,v+1}\right)\right)\right]\end{aligned} \tag{38}$$

The aim of the policy improvement phase is to update the policy to maximize the reward. Based on Ref. [58], to make sure the policy is processable, the Q-value obtained during the policy evaluation phase is first indexed to update the policy. Then, it is converted to the acceptable policy set Π via the minimum Kullback–Leibler divergence. Thus, the update of the policy is defined as follows (39). The loss function of the policy network can be given by (40), in which the parameter φ is updated using stochastic gradients.

$$\begin{aligned}\pi_{\text{new}} &= \text{argmin}_{\pi\in\Pi}D_{\text{KL}}\\ &\left(\pi\left(\cdot|s_{d_i,v}\right)||\frac{\exp\left(\frac{1}{\alpha}\text{Q}_{\pi_{\text{old}}}\left(s_{d_i,v},\cdot\right)\right)}{Z_{\pi_{\text{old}}}\left(s_{d_i,v}\right)}\right)\end{aligned} \tag{39}$$

$$\begin{aligned}\mathcal{L}_\pi(\varphi) &= \mathbb{E}_{s_{d_i,v}\in\mathcal{D}}\sum_{a_{d_i,v}\in\mathcal{A}}\pi_\varphi\left(a_{d_i,v}|s_{d_i,v}\right)\\ &\left(\alpha\log\left(\pi_\varphi\left(a_{d_i,v}|s_{d_i,v}\right)\right)-Q_\xi\left(s_{d_i,v},a_{d_i,v}\right)\right)\end{aligned} \tag{40}$$

Algorithm 2 and Fig. 9 illustrate the training mechanism and pseudo-code of the DCDO-DRL strategy. Algorithm 2 comprises three parts. The first part (lines 1–6) defines initial parameters, including environment critic networks, actor network, target networks, replay buffer, and gradient descent step length. The second part (lines 7–13) interacts with the environment to trigger the action and the next state following the current policy. The transition then is stored in the replay buffer. The third part (lines 14–21) updates the S2S neural network using the stochastic gradients and transitions stored in the replay buffer. The two critic networks, actor network, temperature parameters and two target networks are updated by lines 16–19. Off-policy learning is more effective, mainly because of the ability to learn experience from policies other than the target policy. The core idea of DCDO-DRL strategy in two aspects: (1) the loss function (37) and (40) of the critic and actor networks incorporate an entropy element. (2) The two Q networks as the critic and target networks, respectively. In addition, the loss function (37) and (40) adopt the minimum value of the $Q_\pi(s,a)$ function to improve the training speed.

Algorithm 2 DCDO-DRL

Require: all paraments of the RIDM-TO model

1: Q_{ξ_i} $(i = 1, 2)$ ▷ Initialize critic networks
2: π_φ ▷ Initialize actor networks
3: $Q_{\bar{\xi}_i} \leftarrow Q_{\xi_i}$ $(i = 1, 2)$ ▷ Initialize target networks
4: $\mathcal{D} \leftarrow \emptyset$ ▷ Initialize replay buffer
5: λ_Q, λ_π, λ_α ▷ Step length for gradient descent of the critic network, actor network, temperature parameter
6: $\hat{\mathcal{H}}$ ▷ Initialize minimum value of the average entropy
7: **for** each iteration **do**
8: **for** each environment step **do**
9: $a_{d_i,v} \sim \pi_\varphi\left(a_{d_i,v}|s_{d_i,v}\right)$ ▷ Execute action on current policy
10: $s_{d_i,v+1} \sim p\left(s_{d_i,v+1}|s_{d_i,v}, a_{d_i,v}\right)$ ▷ Transfer to the next state
11: $r_{d_i,v} \leftarrow r(s_{d_i,v}, a_{d_i,v})$ ▷ Receive reward
12: $\mathcal{D} \leftarrow \mathcal{D} \cup \left\{\left(s_{d_i,v}, a_{d_i,v}, r_{d_i,v}, s_{d_i,v+1}\right)\right\}$ ▷ Store transition into $\mathcal{D}$
13: **end for**
14: **for** each gradient step **do**
15: Random sample some $\left(s_{d_i,v}, a_{d_i,v}, r_{d_i,v}, s_{d_i,v+1}\right)$ from $\mathcal{D}$
16: Update two critic networks by $\xi_i = \xi_i - \lambda_Q \nabla_{\xi_i} \mathcal{L}(\xi_i)$ $\forall i \in 1, 2$
17: Update actor network by $\varphi = \varphi - \lambda_\pi \nabla_\varphi \mathcal{L}(\varphi)$
18: Update temperature parameter by $\alpha = \alpha - \lambda_\alpha \nabla_\alpha \mathcal{L}(\alpha)$
19: Update two target networks by $\bar{\xi}_i = \delta\xi_i + (1-\delta)\bar{\xi}_i$ $\forall i \in 1, 2$ ▷ δ is interpolation factor in polyak averaging
20: **end for**
21: **end for**

Ensure: optimal computation offloading strategy π^*

Complexity analysis

The time complexity of DCDO-DRL mainly involves two parts: the S2S neural network and discrete SAC. The S2S neural network includes an encoder (Bi-LSTM), a decoder (LSTM), and an attention mechanism. The time complexity is $\mathcal{O}(L \times N \times M^2)$, $\mathcal{O}(L \times N \times M^2)$, and $\mathcal{O}(L \times N \times M \times H)$, where L is the sequence length, N is the batch size, M is the number of hidden units, H is the number of attention heads. In addition, the time complexity of discrete SAC is $\mathcal{O}(B \times P \times K \times T)$, where B is the batch size, P is the number of parameters in the neural network, K is the number of training steps, and T is the number of computations per step. Hence, the time complexity of DCDO-DRL is the sum of two parts, $\mathcal{O}(L \times N \times (2M^2 + M \times H) + B \times P \times K \times T)$.

Numerical results

This section shows the experimental settings, algorithm convergence, and the impact of attributes on algorithm performance. Furthermore, we investigated the statistical advantages of the DCDO-DRL strategy compared to seven methods in three scenarios.

Simulation setup

To evaluate the performance of the DCDO-DRL strategy, PyCharm is used as the development tool for Python IDE. The S2S neural network is established via the TensorFlow framework. We implement the DCDO-DRL strategy with TensorFlow based on OpenAI Spring Up.

Inspired by Ref. [24], we set the system parameters and initial hyperparameters for the S2S neural network after visiting three centers.[1] The system parameters are given in Table 2, which involve hardware and communication conditions of TD and ES, and RIDM task information. Specifically, the transmission rate and power are set as 7 Mbps and 1.258 W [61]. The CPU computational capacity of TD is 1 G cycles/s, while ES is 9 G cycles/s. We also set the energy coefficients of TD and ES are 1.25×10^{-8} J/cycle and 1.25×10^{-7} J/cycle according to Ref. [61]. In our simulation experiment, it is assumed that the subtasks in the RIDM task are offloaded to ES or TD. The workflow of radiomics is complex and changeable. Hence, we model different RIDM tasks as DAG with different topologies, which elaborate dependencies among modules. The RIDM task data size is set between 250 and 2500 KB and the subtask number V of DAG ranges from 10 to 30 according to the different requirements of radiomics. The computational complexity of each subtask is $10^7 - 10^8$ cycles/s. We select 100 DAGs for each subtask number as the training set and another 20 DAGs as the test set. Then, the S2S neural network is trained based on the information of each subtask in the DAG as input. Finally, to obtain a robust offloading strategy, the DCDO-DRL strategy utilizes discrete SAC to train the S2S neural network.

The S2S neural network is set as a two-layer Bi-LSTM encoder and a two-layer LSTM decoder, each with 256 hidden units and layer normalization [62]. During training, the learning rate is 0.0003, the gradient descent step length is 0.00001, and the batch size is 100. These hyperparameters significantly affect the training and convergence speed of the DCDO-DRL strategy. After initialization and grid search, the optimal hyperparameter settings are presented in Table 3.

Compare algorithms

To evaluate the performance of the DCDO-DRL strategy, we conduct a comparison of the following seven algorithms: (1) local computing (L. Comp.): all subtasks of DAG are executed on the user terminal device without offloading. (2) Full offloading (F. Offl.): all subtasks of DAG are executed on the edge server. (3) Random offloading (R. Offl.): each subtask of the DAG is randomly offloaded to the user terminal device or edge server. (4) Greedy offloading (G. Offl.): find the best

[1] https://www.hnsrmyy.net; https://www.ha.edu.cn; http://ih.ha.edu.cn.

Table 2 The value for system parameters

Parameters	Value	Parameters	Value	Parameters	Value
$b_{d_i,v}$	250–2500 KB	ϕ	$10^7 - 10^8$ cycles/bit	β^t	0.5
$f_{d_i}^l$	1 G cycles/s	$\chi_{d_i}^l$	1.25×10^{-8} J/cycles	β^e	0.5
$f_{d_i}^s$	9–13 G cycles/s	$\chi_{d_i}^s$	1.25×10^{-7} J/cycles	$\Delta f_{e_j}^s$	9–13 G cycles/s
p_{e_j,d_i}^s	1.258 W	r_{e_j,d_i}^s	7 Mbps		

Table 3 The value for the S2S neural network and training hyperparameters

Hyperparameters	Value	Hyperparameters	Value	Hyperparameters	Value	Hyperparameters	Value
Encoder layers	2	Encoder hidden units	Bi-LSTM	Encoder layer type	256	Encoder layer normal	On
Decoder layers	2	Decoder hidden units	LSTM	Decoder layer type	256	Decoder layer normal	On
Learning rate	0.0003	Activation function	Tanh	Optimization method	Adam	Batch size	100
Replay buffer	10,000	γ	0.99	δ	0.995	α	Auto
λ_Q	0.00001	λ_π	0.00001	λ_α	0.00001		

offloading location for each subtask of the DAG by selecting the current optimal solution each time. (5) Round-Robin-based offloading (RR. Offl.): all subtasks of the DAG are alternately performed on the TD and ES. (6) HEFT-based offloading (HEFT. Offl.): HEFT. Offl. [27] adopt the Heterogeneous Earliest Finish Time algorithm to prioritize the subtask of the DAG and run sorting tasks according to the earliest estimated finish time. (7) DRL-based task offloading (DRLTO): DRLTO [24] combined recurrent neural network and DRL to deal with the task offloading scheme and adopt Proximal Policy Optimization to improve the training efficiency.

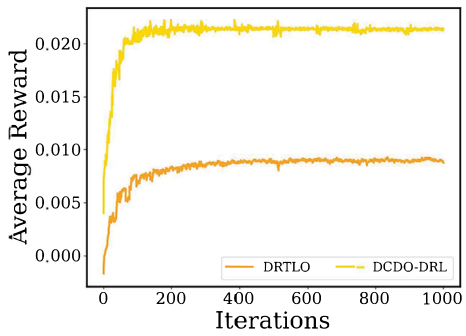


Fig. 10 The average reward of the DCDO-DRL and DRLTO

Convergence analysis

This subsection evaluates the convergence of the proposed DCDO-DRL and DRLTO. The aim of this article is to maximize the RIDM task execution utility. Thus, we set $\beta^t = \beta^e = 0.5$ according to Ref. [24]. The subtask number V of DAG for the RIDM task is 15. The transmission rate is 7 Mbps. The transmission power is 1.258 W. The CPU computational capacity of the main ES and TD is 9 G cycles/s and 1 G cycles/s. Other parameters are detailed in Tables 2 and 3. The DCDO-DRL strategy records the average and updates the S2S neural network at each iteration.

The simulation results are shown in Fig. 10. The x-axis denotes the number of iterations, and the y-axis represents the average reward. It can be noticed that the average reward converges more quickly when the number of iterations is less than 100. As the number of iterations increases, the value grows steadily with a smaller oscillation amplitude. The result demonstrates that the average reward convergence value of the DCDO-DRL strategy is 0.021 at around 200 iterations. Although DRLTO also has the same convergence trend, its convergence speed is lower than the proposed DCDO-DRL strategy in this article. DRLTO converges quickly before 200 iterations and then becomes slower. Finally, the average reward converges to 0.009 in 500 iterations. Therefore, compared to DRLTO, the DCDO-DRL strategy improves training speed. The reason is that the proposed DCDO-DRL strategy maximizes the entropy and the expected reward at the same. This further results in a stronger exploration capability of the DCDO-DRL strategy in the training process.

Impact of subtask numbers

The subsection contrasts the performance of the DCDO-DRL strategy with seven algorithms in terms of various subtask numbers. In this scenario 1, the system is deployed as follows. The subtask number V of DAG for the RIDM task is from 10 to 30. The transmission rate is 7 Mbps. The CPU computational capacity of the main ES and a TD is 9 G cycles/s and 1 G cycles/s. The rest parameter values as shown in Tables 2 and 3. The simulation results are shown in Fig. 11.

By varying the subtask number, The DCDO-DRL strategy has higher utility on the RIDM task compared to the other algorithms. As shown in Fig. 11, the DCDO-DRL strategy has a lower average delay than most algorithms (Fig. 11a). The average energy consumption and average utility are usually lower (Fig. 11b) and higher (Fig. 11c) than those of the other algorithms. When the number of subtasks is small (i.e., N=10), the average delay, average energy consumption and average utility of each algorithm is lower, but the DCDO-DRL strategy still is optimal. As N further increases, all three are increased. The main reason is that increasing the number of subtasks results in a more complex DAG for the RIDM task, which exacerbates the difficulty of task scheduling. In addition, assigning more subtasks to ES reduces the computation time, but also increases the data transmission time, and computation/transmission energy consumption. The computation energy consumption of ES is also higher than TD at the same time. In summary, for the scenario of variable subtask numbers, the DCDO-DRL strategy improved the execution utility of RIDM tasks by 23.07% (computer by $(\frac{\text{utility}_{\text{DCDO-DRL}}-\text{utility}_{\text{DRLTO}}}{\text{utility}_{\text{DRLTO}}})$ compared to DRLTO.

In addition, we analyze the correlation between average delay, average energy consumption, and average utility. The joint distribution diagram is a visual representation to display the interrelationship between the two variables. Figure 12a demonstrates the joint distribution between the average delay and average utility under scenario 1. It can be seen from the regression line that the two variables have a positive correlation. As the average delay increases, the average utility shows an increasing trend. Figure 12b shows the joint distribution between the average energy consumption and average utility under scenario 1. The regression line also exhibits that there is also a positive correlation.

Impact of transmission rate

The subsection contrasts the performance of the DCDO-DRL strategy on various transmission rates with seven algorithms. In this scenario 2, the system is deployed as follows. The transmission rate ranges from 5 Mbps to 17 Mbps. The subtask number V of DAG for the RIDM task is 15. The CPU computational capacity of the main ES and TD also is 9 G cycles/s and 1 G cycles/s. The rest parameters as shown in Tables 2 and 3. The simulation results are shown in Fig. 13.

By varying the transmission rate, the results show that the DCDO-DRL strategy has better performance on the RIDM task than other algorithms. As shown in Fig. 13, the average delay and average energy consumption of L. Offl. are fixed. When the transmission rate is small (i.e., $r^s_{e_j,d_i} = 5$), transmitting all data to ES incurs considerable delays (Fig. 13a). When $r^s_{e_j,d_i} = 7, 9$, L. Offl. has the lowest average energy consumption and the highest average delay (Fig. 13b). As $r^s_{e_j,d_i}$ further increases, the average delay and average energy consumption of all algorithms decrease (except for L. Offl.). The DCDO-DRL strategy has the lowest energy consumption (Fig. 13a, b). This is because the higher transmission rate is beneficial for offloading tasks to ES. Figure 13c shows the average utility of F. Offl. – DCDO-DRL gradually increases as the transmission rate increases. This is because the reduction in transmission time drives the execution of subtasks on ES. To sum up, the DCDO-DRL strategy improved the execution utility of the RIDM task by 12.77% compared to the suboptimal DRLTO, specifically in scenarios related to transmission rates.

Similarly, the histogram on the upper and right sides of Fig. 14a demonstrates the marginal distribution of average delay and average utility under scenario 2, respectively. The middle part shows the joint distribution between the two variables. The histogram on the upper of Fig. 14b displays the marginal distribution of average energy consumption. The two regression lines in Fig. 14 show a negative slope, implying a negative correlation between all average latency and average energy consumption and average utility. The shaded part shows the confidence interval of the regression lines. As the increase of two variables, the average utility displays a decreasing trend. However, it can be clearly seen that the regression line in Fig. 14a is steeper compared to Fig. 14b. Therefore, the average delay has a greater effect on the average utility.

Impact of CPU computational capacity

To further evaluate the DCDO-DRL strategy, this subsection compares the performance of the DCDO-DRL strategy on various CPU computational capabilities with seven algorithms. In this scenario 3, the system is deployed as follows. The CPU computational capacity of the main ES ranges from 1 G cycles/s to 8 G cycles/s. The transmission rate is 7 Mbps. The subtask number V of DAG for the RIDM task is 15. The CPU computational capacity of TD is 1 G cycles/s. The rest parameter values as shown in Tables 2 and 3. The simulation results are shown in Fig. 15.

By adjusting the computing power of ES, the DCDO-DRL strategy performs better in the RIDM task. As shown in Fig. 15, the average delay, average energy consumption and average utility of L. Offl. are constant. The reason is that L. Offl. is not affected by the computing power of ES. When the computational power of ES is small (i.e., $f^s_{e_j} = 1$), it is equivalent to that of TD. Running all data on ES will generate massive energy consumption. Thus, the energy consumption of F. Offl. is huge in Fig. 15b. When $f^s_{e_j} = 2$, the average energy consumption and the average utility of the F. Offl. drops abruptly and increases steeply, respectively. As $f^s_{e_j}$ is further increased, there is little difference in the average delay of the individual algorithms (Fig. 15a). The average energy

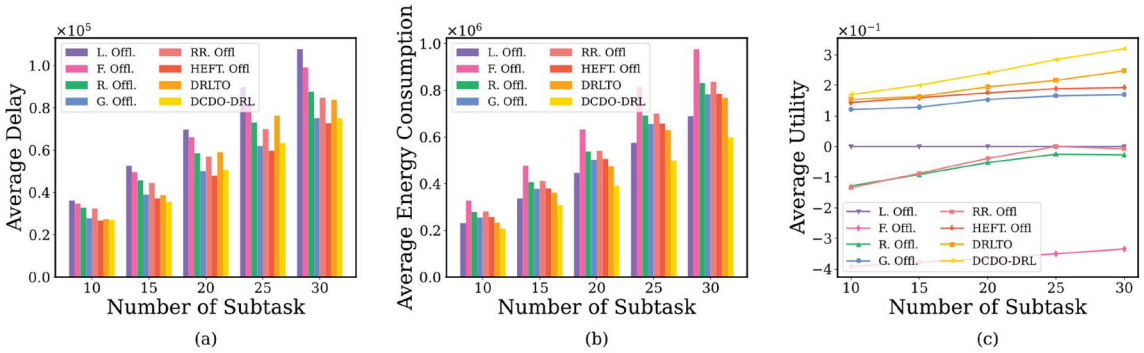
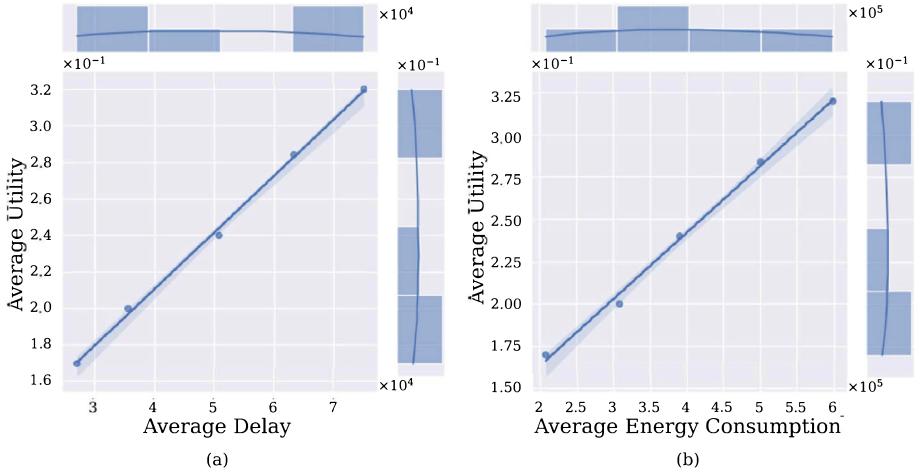


Fig. 11 Illustration on the impact of subtask number

Fig. 12 Joint distribution between average utility and average delay or average energy consumption under scenario 1



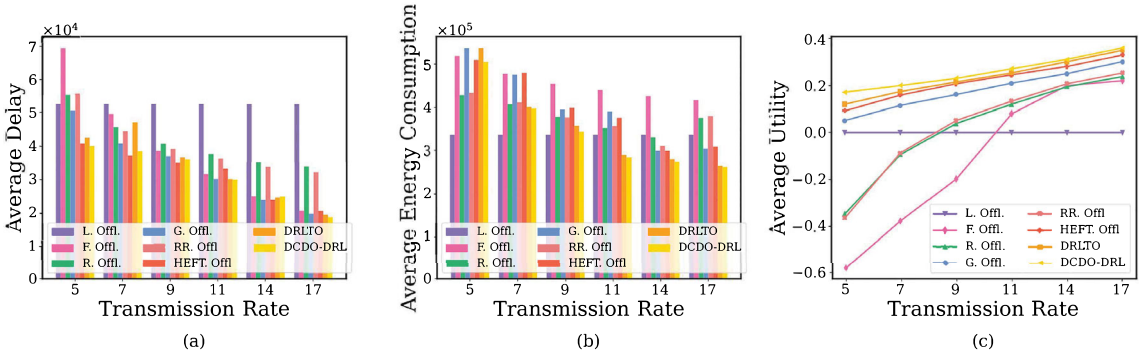


Fig. 13 Illustration on the impact of the transmission rate

consumption shows a steadily decreasing trend, while the average utility increases slowly except for L. Offl. and F. Offl. (Fig. 15b, c). The influence of ES's computing capability gradually becoming smaller is the primary cause of this. In conclusion, compared to the suboptimal DRLTO, the DCDO-DRL strategy improves the execution efficiency of RIDM tasks by 8.51% when faced with different CPU computing power.

Likewise, Fig. 16 demonstrates the joint distribution between the average delay, average energy consumption and average utility under scenario 3. The result shows that both regression lines show a negative correlation between the two variables. However, compared to the regression line with Fig. 16a, the one with Fig. 16b is steeper. This reflects the fact that average energy consumption has a greater impact on average utility.

Statistical superiority analysis

Statistical test is a widely used method to evaluate the performance of the algorithm in various fields. In the above analysis, we compute the average utility performance of

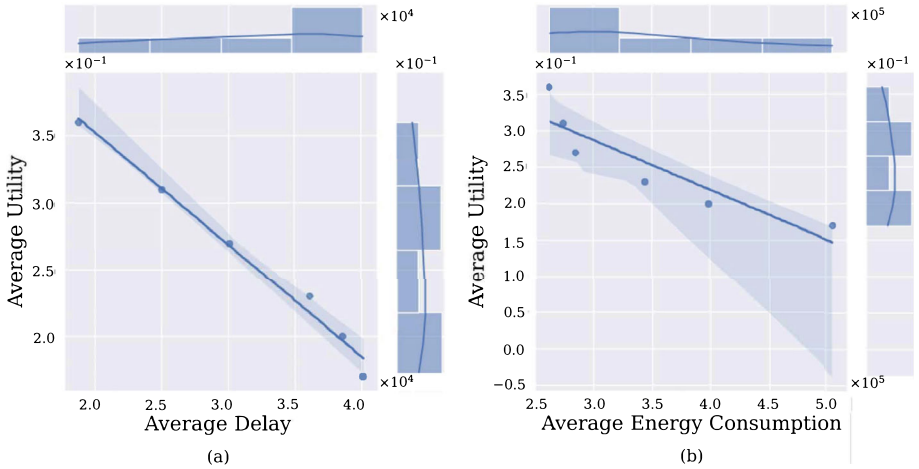


Fig. 14 Illustration on joint distribution between average utility and average delay or average energy consumption under scenario 2

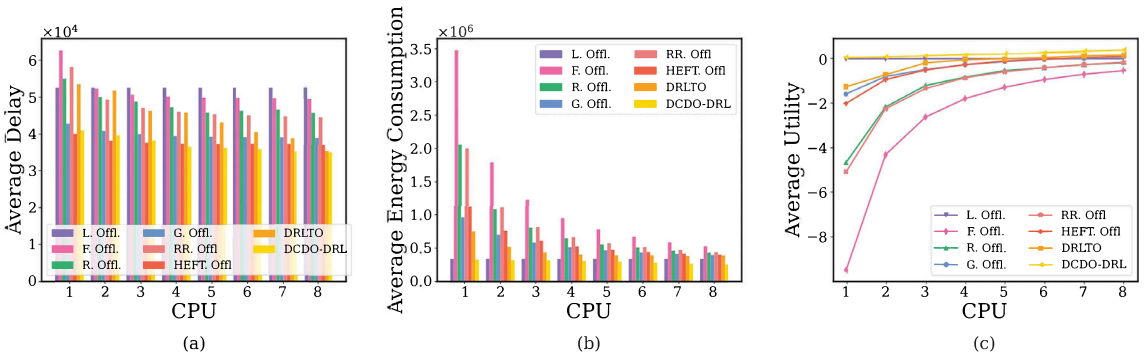


Fig. 15 Illustration on the impact of CPU computational capacity

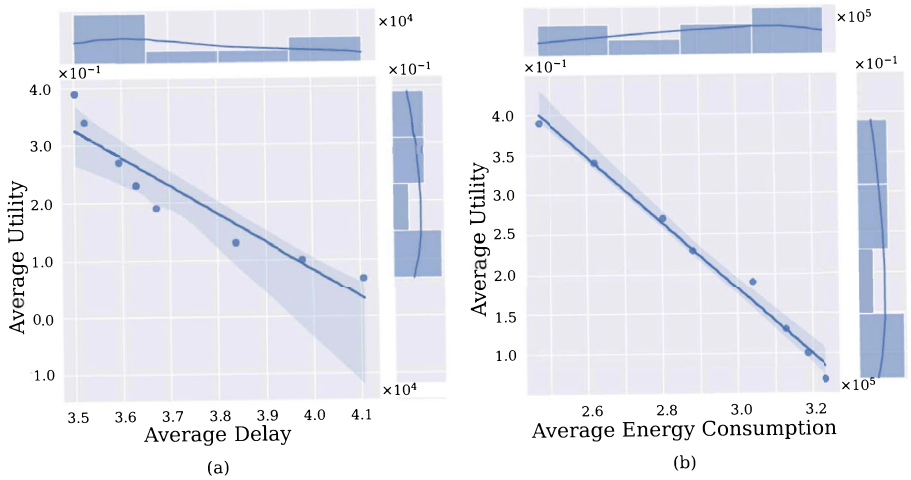


Fig. 16 Illustration on joint distribution between average utility and average delay or average energy consumption under scenario 3

seven algorithms across different subtask numbers, different transmission rates, and different CPU computational capacities. To determine the superiority of the DCDO-DRL strategy in three scenarios, we conduct pairwise comparisons. In this article, we use the Wilcoxon rank sum test [60] as a non-parametric statistical test. The test compares the significant level differences between algorithms by P value. Note that we consider the algorithm to be statistically different if and only if the P value is less than 0.05. The P values calculated under three scenarios are shown in Table 4. For the subtask numbers, the P values are less than 0.05, indicating a statistically significant difference between the DCDO-DRL strategy and other algorithms. Similarly, there is a statistically significant difference in transmission rate, as the P values are all less than 0.05. In terms of CPU computing power, although not all P values are less than 0.05, 5 out of 7 also shows statistically significant differences. To sum up, by counting the P values of DCDO-DRL and seven algorithms, it can be

Table 4 The P value under various experiment settings

Aspects	DCDO-DRL V.S L. Offl.	DCDO-DRL V.S F. Offl.	DCDO-DRL V.S R. Offl.	DCDO-DRL V.S G. Offl.	DCDO-DRL V.S RR. Offl.	DCDO-DRL V.S HEFT. Offl.	DCDO-DRL V.S DRLTO
Subtask number	0.0037	0.0060	0.0060	0.0079	0.0060	0.0300	0.0472
Transmission rate	0.0013	0.0123	0.0153	0.0463	0.0226	**0.0892**	**0.1481**
CPU	0.0002	0.0004	0.0004	0.0006	0.0004	0.0009	0.0037

found that only two of the 21 P values exceed 0.05, reflecting the statistical superiority of the DCDO-DRL strategy in maximizing the RIDM task execution utility. To sum up, by counting the P values of DCDO-DRL against the seven algorithms, it can be found that only two of the 21 P values exceed 0.05, reflecting the statistical superiority of the DCDO-DRL strategy in maximizing the RIDM task execution utility.

Conclusion

In this article, we propose a DCDO-DRL strategy, which plays a significant role in improving the RIDM execution efficiency and adapting to the different RIDM environments in the medical image cloud. DCDO-DRL aims to maximize the RIDM task utility, a weighted sum of DEC generated by execution. Specifically, the internal dependencies of the RIDM task based on radiomics are modeled by a DAG. The offloading decision process in the DAG is represented by the sequence prediction of the S2S neural network. Next, we propose a DCP algorithm to accelerate subtask processing by collaborating with multiple ES resources. Finally, to improve the robustness of the S2S neural network, the DCDO-DRL strategy follows the discrete SAC. The results show the execution utility of the DCDO-DRL strategy in the RIDM task by at least 23.07, 12.77, and 8.51% in three scenarios.

It is worth noting that content caching is also an effective way to decrease computational delay and energy consumption. Therefore, our future research work focuses on exploring the problem of combining content caching and task offloading. One potential solution is to formulate the problem as a mixed-integer non-linear programming (MINLP) problem. Then, the MINLP problem is then proved to be a 0–1 knapsack problem and solved by an efficient algorithm.

Acknowledgements This work was supported by the National Natural Science Foundation of China (No. 81772009), the Collaborative Innovation Major Project of Zhengzhou (No. 20XTZX06013, No. 20XTZX05015), and the Key Technologies R&D Program of Henan Province (No. 212102310039).

Data availability We used simulation data in our experiment, not publicly available datasets.

Declarations

Conflict of interest The authors declare that they have no conflict of interest.

Appendix

Definition 1 RIDM task offloading system model is a 12-tuple: RIDM-TO $= (S, D, \mathrm{DCP}, G, B, \mu, \phi, \zeta, V^l, V^s, T^{total}, \Psi^{total})$, where:

(1) $E = \left\{e_j | j = 1, 2, \ldots, m\right\}$ is the finite set of edge servers. m refers to the total number of edge servers, $m \in \mathbb{N}^+$.
(2) $D = \{d_i | i = 1, 2, \ldots, n\}$ is the finite set of terminal devices, i.e., PC device used by the user. n refers to the total number of users, $n \in \mathbb{N}^+$.
(3) DCP is the distributed collaborative processing algorithm on the edge layer.
(4) $G = \left\{g_{d_i} | i = 1, 2, \ldots, n\right\}$ is the finite set of DAG topologies modeled of RIDM. (See Def. 2).
(5) $B = \left\{b_{d_i,v} = \left(b^l_{d_i,v}, b^u_{d_i,v}, b^c_{d_i,v}, b^d_{d_i,v}\right) | i = 1,2,\ldots,n; v = 1, 2, \ldots, V\right\}$ is the finite set of the RIDM task data sizes.
(6) $\mu = \left\{\mu_{d_i} = \left\{\mu_{d_i,v}\right\} | i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of offloading strategies for terminal devices. V is the number of subtasks.
(7) ϕ is task computational complexity, i.e., Required CPU cycles for computing per bit of $b_{d_i,v}$. The computational complexity of each subtask is the same.
(8) ζ : TD $\longrightarrow$ ES is the mapping from TD to ES, meaning the TD is covered by ES in the communications area.
(9) V^l is the local computing parameters (See Def. 3).
(10) V^s is the edge collaboration computing parameters (See ***Definition 4***).
(11) $T^{total} = \left\{\tau^{total}_{d_i} | i = 1, 2, \ldots, n\right\}$ is the finite set of the total delays for DAGs of the terminal device.
(12) $\Psi^{total} = \left\{\psi^{total}_{d_i} | i = 1, 2, \ldots, n\right\}$ is the finite set of the total energy consumptions for DAGs of the terminal device.

Definition 3 Local computing parameters model is 4-tuple: $V^l = \left(F^l, \chi^l, T^l, \Psi^{l,c}\right)$, where:

(1) $F^l = \left\{f^l_{d_i} | i = 1, 2, \ldots, n\right\}$ is the finite set of the computational capabilities of terminal devices.
(2) $\chi^l = \left\{\chi^l_{d_i} | i = 1, 2, \ldots, n\right\}$ is the finite set of the energy coefficients of terminal devices.
(3) $T^l = \left(T^{l,c}, m^{l,c}, \mathrm{FT}^{l,c}\right)$ is the finite set of local delay parameters, where:
   (a) $T^{l,c} = \left\{\tau^{l,c}_{d_i,v} | i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the computation delay locally.
   (b) $\mathrm{ST}^{l,c} = \left\{\mathrm{st}^{l,c}_{d_i,v} | i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the actual execution start time locally.
   (c) $\mathrm{FT}^{l,c} = \left\{\mathrm{ft}^{l,c}_{d_i,v} | i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the actual execution finish time locally.
(4) $\Psi^{l,c} = \left\{\psi^{l,c}_{d_i,v} | i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the computation energy consumption locally.

Definition 4 Edge collaboration computing parameters model is 10-tuple: $V^s = (F^s, \chi^s, \Delta M^s, \Delta F^s, P^s, R^s, T^u, T^c, T^d, E^s)$, where:

(1) $F^s = \left\{f^s_{e_j} | j = 1, 2, \ldots, m\right\}$ is the finite set of the computational capabilities of edge servers.
(2) $\chi^s = \left\{\chi^s_{e_j} | j = 1, 2, \ldots, m\right\}$ is the finite set of the energy coefficients of edge servers.
(3) $\Delta M^s = \left\{\Delta m^s_{e_j} | j = 1, 2, \ldots, m\right\}$ is the finite set of the residual memories of edge servers.
(4) $\Delta F^s = \left\{\Delta f^s_{e_j} | j = 1, 2, \ldots, m\right\}$ is the finite set of the residual computational capabilities of edge servers.
(5) $P^s = \left\{p^s_{e_j,d_i} | j = 1, 2, \ldots, m; i = 1, 2, \ldots, n\right\}$ is the finite set of transmission powers.
(6) $R^s = \left\{r^s_{e_j,d_i} | j = 1, 2, \ldots, m; i = 1, 2, \ldots, n\right\}$ is the finite set of the data transmission rates.
(7) $T^u = (T^{s,u}, \mathrm{ST}^{s,u}, \mathrm{FT}^{s,u})$ is the finite set of delay parameters on the uplink channel, where:
   (a) $T^{s,u} = \left\{\tau^{s,u}_{e_j,d_i,v} | j = 1, 2, \ldots, m; i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the transmission delay via the uplink channel.
   (b) $\mathrm{ST}^{s,u} = \left\{\mathrm{st}^{s,u}_{e_j,d_i,v} | j = 1, 2, \ldots, m; i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the actual execution start time to e_j via the uplink channel.
   (c) $\mathrm{FT}^{s,u} = \left\{\mathrm{ft}^{s,u}_{e_j,d_i,v} | j = 1, 2, \ldots, m; i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the actual execution finish time to e_j via the uplink channel.
(8) $T^c = (T^{s,c}, \mathrm{ST}^{s,c}, \mathrm{FT}^{s,c})$ is the finite set of delay parameters on the edge server, where:
   (a) $T^{s,c} = \left\{\tau^{s,c}_{e_j,d_i,v} | j = 1, 2, \ldots, m; i = 1, 2, \ldots, n; v = 1, 2, \ldots, V\right\}$ is the finite set of the computation delay of executing subtask on e_j.

(b) $\mathrm{ST}^{s,c} = \left\{ \mathrm{st}^{s,c}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the actual execution start time on e_j.

(c) $\mathrm{FT}^{s,c} = \left\{ \mathrm{ft}^{s,c}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the actual execution finish time on e_j.

(9) $T^d = \left(T^{s,d}, \mathrm{ST}^{s,d}, \mathrm{FT}^{s,d}\right)$ is the finite set of delay parameters on the downlink channel, where:

(a) $T^{s,d} = \left\{ \tau^{s,d}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the transmission delay to e_j via the downlink channel.

(b) $\mathrm{ST}^{s,d} = \left\{ \mathrm{st}^{s,d}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the actual execution start time to e_j via the downlink channel.

(c) $\mathrm{FT}^{s,d} = \left\{ \mathrm{ft}^{s,d}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the actual execution finish time to e_j via the downlink channel.

(10) $E^s = \left(\Psi^{s,u}, \Psi^{s,c}, \Psi^{s,d}\right)$ is the finite set of energy consumption parameters on ECCM, where:

(a) $\Psi^{s,u} = \left\{ \psi^{s,u}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the transmission energy consumption to e_j via the uplink channel.

(b) $\Psi^{s,c} = \left\{ \psi^{s,c}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the computation energy consumption on e_j.

(c) $\Psi^{s,d} = \left\{ \psi^{s,d}_{e_j,d_i,v} \mid j = 1,2,\ldots,m; i = 1,2,\ldots,n; v = 1,2,\ldots,V \right\}$ is the finite set of the transmission energy consumption to e_j via the downlink channel.

References

1. Wang X, Zhang Y, Guo Z, Li J (2021) TMRGM: a template-based multi-attention model for X-ray imaging report generation. J Artif Intell Med Sci 2(1):21–32. https://doi.org/10.2991/jaims.d.210428.002
2. Mao Q, Zhou MT, Zhao ZP, Liu N, Yang L, Zhang XM (2022) Role of radiomics in the diagnosis and treatment of gastrointestinal cancer. World J Gastroenterol 28(42):6002–6016. https://doi.org/10.3748/wjg.v28.i42.6002
3. Lakshmi C, Thenmozhi K, Rayappan JBB, Rajagopalan S, Amirtharajan R, Chidambaram N (2021) Neural-assisted image-dependent encryption scheme for medical image cloud storage. Neural Comput Appl 33(12):6671–6684. https://doi.org/10.1007/s00521-020-05447-9
4. Qin X, Li B, Ying L (2023) Efficient distributed threshold-based offloading for large-scale mobile cloud computing. IEEEACM Trans Netw 31(1):308–321. https://doi.org/10.1109/TNET.2022.3193073
5. Liu T, Fang L, Zhu Y, Tong W, Yang Y (2022) A near-optimal approach for online task offloading and resource allocation in edge-cloud orchestrated computing. IEEE Trans Mob Comput 21(8):2687–2700. https://doi.org/10.1109/TMC.2020.3045471
6. Lin R, Guo X, Luo S, Xiao Y, Moran B, Zukerman M (2023) Application-aware computation offloading in edge computing networks. Future Gener Comp Syst 146:86–97. https://doi.org/10.1016/j.future.2023.04.009
7. Khoobkar MH, Dehghan Takht Fooladi M, Rezvani MH, Gilanian Sadeghi MM (2023) Joint optimization of delay and energy in partial offloading using dual-population replicator dynamics. Expert Syst Appl 216:119417. https://doi.org/10.1016/j.eswa.2022.119417
8. Chen R, Wang X (2023) Maximization of value of service for mobile collaborative computing through situation aware task offloading. IEEE Trans Mob Comput 22(2):1049–1065. https://doi.org/10.1109/TMC.2021.3086687
9. Liu J, Ren J, Zhang Y, Peng X, Zhang Y, Yang Y (2023) Efficient dependent task offloading for multiple applications in MEC-cloud system. IEEE Trans Mob Comput 22(4):2147–2162. https://doi.org/10.1109/TMC.2021.3119200
10. Duan S, Wang D, Ren J, Lyu F, Zhang Y, Wu H, Shen X (2023) Distributed artificial intelligence empowered by end-edge-cloud computing: a survey. IEEE Commun Surv Tutor 25(1):591–624. https://doi.org/10.1109/COMST.2022.3218527
11. Kwok YK, Ahmad I (1999) Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Comput Surv 31(4):406–471. https://doi.org/10.1145/344588.344618
12. Ma Z, Zhang S, Chen Z, Han T, Qian Z, Xiao M, Chen N, Wu J, Lu S (2022) Towards revenue-driven multi-user online task offloading in edge computing. IEEE Trans Parallel Distrib Syst 33(5):1185–1198. https://doi.org/10.1109/TPDS.2021.3105325
13. Tong Z, Wang J, Mei J, Li K, Li W, Li K (2023) Multi-type task offloading for wireless internet of things by federated deep reinforcement learning. Future Gener Comp Syst 145:536–549. https://doi.org/10.1016/j.future.2023.04.004
14. Zhang Z, Cui P, Zhu W (2022) Deep learning on graphs: a survey. IEEE Trans Knowl Data Eng 34(1):249–270. https://doi.org/10.1109/TKDE.2020.2981333
15. Haarnoja T, Zhou A, Hartikainen K, Tucker G, Ha S, Tan J, Kumar V, Zhu H, Gupta A, Abbeel P, Levine S (2019) Soft actor-critic algorithms and applications. https://doi.org/10.48550/arXiv.1812.05905
16. de Freitas Cunha RL, Chaimowicz L (2023) An SMDP approach for reinforcement learning in HPC cluster schedulers. Future Gener Comp Syst 139:239–252. https://doi.org/10.1016/j.future.2022.09.025
17. Demir S (2022) Turkish data-to-text generation using sequence-to-sequence neural networks. ACM Trans Asian Low-Resour Lang Inf Process 22(2):37-1–37-27. https://doi.org/10.1145/3543826
18. Zhang Y, Qiu M, Tsai CW, Hassan MM, Alamri A (2017) Health-CPS: healthcare cyber-physical system assisted by cloud and big data. IEEE Syst J 11(1):88–95. https://doi.org/10.1109/JSYST.2015.2460747
19. Khezr SN, Navimipour NJ (2017) MapReduce and its applications, challenges, and architecture: a comprehensive review and directions for future research. J Grid Comput 15(3):295–321. https://doi.org/10.1007/s10723-017-9408-0
20. Mo Y (2019) A data security storage method for IoT under hadoop cloud computing platform. Int J Wirel Inf Netw 26(3):152–157. https://doi.org/10.1007/s10776-019-00434-x
21. Duan Y, Edwards JS, Dwivedi YK (2019) Artificial intelligence for decision making in the era of big data—evolution, challenges and research agenda. Int J Inf Manag 48:63–71. https://doi.org/10.1016/j.ijinfomgt.2019.01.021
22. Rahman MS, Khalil I, Yi X (2019) A lossless DNA data hiding approach for data authenticity in mobile cloud based healthcare

systems. Int J Inf Manag 45:276–288. https://doi.org/10.1016/j.ijinfomgt.2018.08.011
23. El-Seoud SA, El-Sofany HF, Abdelfattah MAF, Mohamed R (2017) Big data and cloud computing: trends and challenges. Int J Interact Mob Technol 11(2):34–52. https://doi.org/10.3991/ijim.v11i2.6561
24. Wang J, Hu J, Min G, Zhan W, Zomaya AY, Georgalas N (2022) Dependent task offloading for edge computing based on deep reinforcement learning. IEEE Trans Comput 71(10):2449–2461. https://doi.org/10.1109/TC.2021.3131040
25. Li H, Xiong K, Lu Y, Gao B, Fan P, Letaief K (2023) Distributed design of wireless powered fog computing networks with binary computation offloading. IEEE Trans Mob Comput 22(4):2084–2099. https://doi.org/10.1109/TMC.2021.3115348
26. Pan Y, Pan C, Wang K, Zhu H, Wang J (2021) Cost minimization for cooperative computation framework in MEC networks. IEEE Trans Wirel Commun 20(6):3670–3684. https://doi.org/10.1109/TWC.2021.3052887
27. Zhang Y, Chen J, Zhou Y, Yang L, He B, Yang Y (2022) Dependent task offloading with energy-latency tradeoff in mobile edge computing. IET Commun 16(17):1993–2001. https://doi.org/10.1049/cmu2.12454
28. Fu S, Zhou F, Hu RQ (2022) Resource allocation in a relay-aided mobile edge computing system. IEEE Internet Things J 9(23):23659–23669. https://doi.org/10.1109/JIOT.2022.3190470
29. Bi J, Yuan H, Zhang K, Zhou M (2022) Energy-minimized partial computation offloading for delay-sensitive applications in heterogeneous edge networks. IEEE Trans Emerg Top Comput 10(4):1941–1954. https://doi.org/10.1109/TETC.2021.3137980
30. Wang Z, Jia Z, Liao H, Zhou Z, Zhao X, Zhang L, Mumtaz S, Rodrigues JJPC (2020) Energy-aware and URLLC-aware task offloading for internet of health things. In: GLOBECOM 2020—2020 IEEE Global Communications Conference, pp 1–6. https://doi.org/10.1109/GLOBECOM42002.2020.9348237
31. Seid AM, Boateng GO, Mareri B, Sun G, Jiang W (2021) Multi-agent DRL for task offloading and resource allocation in multi-UAV enabled IoT edge network. IEEE Trans Netw Serv Manag 18(4):4531–4547. https://doi.org/10.1109/TNSM.2021.3096673
32. Alam MZ, Jamalipour A (2022) Multi-agent DRL-based Hungarian algorithm for task offloading in multi-access edge computing internet of vehicles. IEEE Trans Wirel Commun 21(9):7641–7652. https://doi.org/10.1109/TWC.2022.3160099
33. Zhan Y, Guo S, Li P, Zhang J (2020) A deep reinforcement learning based offloading game in edge computing. IEEE Trans Comput 69(6):883–893. https://doi.org/10.1109/TC.2020.2969148
34. Chen S, Chen J, Miao Y, Wang Q, Zhao C (2022) Deep reinforcement learning-based cloud-edge collaborative mobile computation offloading in industrial networks. IEEE Trans Signal Inf Process Netw 8:364–375. https://doi.org/10.1109/TSIPN.2022.3171336
35. Wang X, Ning Z, Guo L, Guo S, Gao X, Wang G (2023) Mean-field learning for edge computing in mobile blockchain networks. IEEE Trans Mob Comput 22(10):5978–5994. https://doi.org/10.1109/TMC.2022.3186699
36. Shi J, Du J, Shen Y, Wang J, Yuan J, Han Z (2023) DRL-based V2V computation offloading for blockchain-enabled vehicular networks. IEEE Trans Mob Comput 22(7):3882–3897. https://doi.org/10.1109/TMC.2022.3153346
37. Tutsoy O (2022) Pharmacological, non-pharmacological policies and mutation: an artificial intelligence based multi-dimensional policy making algorithm for controlling the casualties of the pandemic diseases. IEEE Trans Pattern Anal Mach Intell 44(12):9477–9488. https://doi.org/10.1109/TPAMI.2021.3127674
38. Li Y, Wei D, Liu X, Fan X, Wang K, Li S, Zhang Z, Ma K, Qian T, Jiang T, Zheng Y, Wang Y (2022) Molecular subtyping of diffuse gliomas using magnetic resonance imaging: comparison and correlation between radiomics and deep learning. Eur Radiol 32(2):747–758. https://doi.org/10.1007/s00330-021-08237-6
39. Liu Z, Wang S, Dong D, Wei J, Fang C, Zhou X, Sun K, Li L, Li B, Wang M, Tian J (2019) The applications of radiomics in precision diagnosis and treatment of oncology: opportunities and challenges. Theranostics 9(5):1303–1322. https://doi.org/10.7150/thno.30309
40. Scapicchio C, Gabelloni M, Barucci A, Cioni D, Saba L, Neri E (2021) A deep look into radiomics. Radiol Med 126(10):1296–1311. https://doi.org/10.1007/s11547-021-01389-x
41. Yu J, Li F, Hu X (2023) Two-stage decolorization based on histogram equalization and local variance maximization. SIAM J Imaging Sci 16(2):740–769. https://doi.org/10.1137/22M1509333
42. Bhardwaj R (2023) Hiding patient information in medical images: an enhanced dual image separable reversible data hiding algorithm for e-healthcare. J Ambient Intell Humaniz Comput 14(1):321–337. https://doi.org/10.1007/s12652-021-03299-2
43. Liu Y, Wang W, Li Y, Lai H, Huang S, Yang X (2023) Geometry-consistent adversarial registration model for unsupervised multi-modal medical image registration. IEEE J Biomed Health Inform 27(7):3455–3466. https://doi.org/10.1109/JBHI.2023.3270199
44. Xia L, Zhang H, Wu Y, Song R, Ma Y, Mou L, Liu J, Xie Y, Ma M, Zhao Y (2022) 3d vessel-like structure segmentation in medical images by an edge-reinforced network. Med Image Anal 82:102581. https://doi.org/10.1016/j.media.2022.102581
45. Reena Roy R, Anandha Mala GS (2023) An improved k-means clustering for segmentation of pancreatic tumor from CT images. IETE J Res 69(7):3966–3973. https://doi.org/10.1080/03772063.2021.1944335
46. Wang C, Pedrycz W, Li Z, Zhou M (2021) Residual-driven fuzzy c-means clustering for image segmentation. IEEE/CAA J Autom Sinica 8(4):876–889. https://doi.org/10.1109/JAS.2020.1003420
47. Shahdoosti HR, Javaheri N (2018) A fast algorithm for feature extraction of hyperspectral images using the first order statistics. Multimed Tools Appl 77(18):23633–23650. https://doi.org/10.1007/s11042-018-5695-0
48. Jindal B, Garg S (2023) FIFE: fast and indented feature extractor for medical imaging based on shape features. Multimed Tools Appl 82(4):6053–6069. https://doi.org/10.1007/s11042-022-13589-2
49. Chunmei X, Mei H, Yan Z, Haiying W (2019) Diagnostic method of liver cirrhosis based on MR image texture feature extraction and classification algorithm. J Med Syst 44(1):11. https://doi.org/10.1007/s10916-019-1508-x
50. Kumar Singh V, Kalafi EY, Wang S, Benjamin A, Asideu M, Kumar V, Samir AE (2022) Prior wavelet knowledge for multi-modal medical image segmentation using a lightweight neural network with attention guided features. Expert Syst Appl 209:118166. https://doi.org/10.1016/j.eswa.2022.118166
51. Cheng C, Hua ZC (2020) Lasso peptides: heterologous production and potential medical application. Front Bioeng Biotechnol 8:571165. https://doi.org/10.3389/fbioe.2020.571165
52. Li BQ, Huang T, Liu L, Cai YD, Chou KC (2012) Identification of colorectal cancer related genes with mRMR and shortest path in protein–protein interaction network. PLoS One 7(4):e33393. https://doi.org/10.1371/journal.pone.0033393
53. Ma R, Cai TT, Li H (2021) Global and simultaneous hypothesis testing for high-dimensional logistic regression models. J Am Stat Assoc 116(534):984–998. https://doi.org/10.1080/01621459.2019.1699421
54. Cortes C, Vapnik V (1995) Support-vector networks. Mach Learn 20(3):273–297. https://doi.org/10.1023/A:1022627411411
55. Chen X, Jiao L, Li W, Fu X (2016) Efficient multi-user computation offloading for mobile-edge cloud computing. IEEEACM Trans Netw 24(5):2795–2808. https://doi.org/10.1109/TNET.2015.2487344
56. Shi X, Zhang X, Zhuang F, Lu Y, Liang F, Zhao N, Wang X, Li Y, Cai Z, Wu Z, Shen L, He B (2022) Congestive heart failure detection

based on attention mechanism-enabled bi-directional long short-term memory model in the internet of medical things. J Ind Inf Integr 30:100402. https://doi.org/10.1016/j.jii.2022.100402
57. Amin SU, Altaheri H, Muhammad G, Abdul W, Alsulaiman M (2022) Attention-inception and long- short-term memory-based electroencephalography classification for motor imagery tasks in rehabilitation. IEEE Trans Ind Inform 18(8):5412–5421. https://doi.org/10.1109/TII.2021.3132340
58. Haarnoja T, Zhou A, Abbeel P, Levine S (2018) Soft actor-critic: off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: Dy J, Krause A (eds) Proceedings of the 35th international conference on machine learning, PMLR, vol 80, pp 1861–1870. https://doi.org/10.48550/arXiv.1801.01290
59. Christodoulou P (2019) Soft actor-critic for discrete action settings. https://doi.org/10.48550/arXiv.1910.07207
60. Derrac J, García S, Molina D, Herrera F (2011) A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. Swarm Evol Comput 1(1):3–18. https://doi.org/10.1016/j.swevo.2011.02.002
61. Thinh TQ, Tang J, La QD, Quek TQS (2017) Offloading in mobile edge computing: task allocation and computational frequency scaling. IEEE Trans Commun 65(8):3571–3584. https://doi.org/10.1109/TCOMM.2017.2699660
62. Ba JL, Kiros JR, Hinton GE (2016) Layer normalization. https://doi.org/10.48550/arXiv.1607.06450