Journal of Systems Architecture 154 (2024) 103215

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

UAV-assisted dependency-aware computation offloading in device–edge–cloud collaborative computing based on improved actor–critic DRL

Longxin Zhang [a], Runti Tan [a], Yanfen Zhang [a], Jiwu Peng [b,*], Jing Liu [c], Keqin Li [d]

[a] College of Computer Science, Hunan University of Technology, Zhuzhou 412007, China
[b] College of Information Technology and Management, Hunan University of Finance and Economics, Changsha, 410205, China
[c] Department Computer Science and Technology, Wuhan University of Science and Technology, Wuhan 430000, China
[d] Department of Computer Science, State University of New York, New Paltz, NY 12561, USA

ARTICLE INFO

Dataset link: https://github.com/TanRunti/JSA-STSUDCO

Keywords:
Computation offloading
Device–edge–cloud collaboration
Dependent application
Soft actor–critic
UAV assistance

ABSTRACT

Unmanned aerial vehicle (UAV)-assisted mobile edge computing (MEC) has become a popular research topic, addressing challenges posed by the pressure of cloud computing and the limited service scope of MEC. However, the limited computing resources of UAVs and the data dependency of specific tasks hinder the practical implementation of efficient computational offloading (CO). Accordingly, a device–edge–cloud collaborative computing model is proposed in this study to provide complementary offloading services. This model considers stochastic movement and channel obstacles, representing the dependency relationships as a directed acyclic graph. An optimization problem is formulated to simultaneously optimize system costs (i.e., delay and energy consumption) and UAV endurance, taking into account resource and task-dependent constraints. Additionally, a saturated training SAC-based UAV-assisted dependency-aware computation offloading algorithm (STS-UDCO) is developed. STS-UDCO learns the entropy and value of the CO policy to efficiently approximate the optimal solution. The adaptive saturation training rule proposed in STS-UDCO dynamically controls the update frequency of the critic based on the current fitted state to enhance training stability. Finally, extensive experiments demonstrate that STS-UDCO achieves superior convergence and stability, while also reducing the system total cost and convergence speed by at least 11.83% and 39.10%, respectively, compared with other advanced algorithms.

1. Introduction

In recent years, traditional cloud computing platforms have encountered significant challenges arising from the development of the Internet of Things and next-generation communication technologies. These advancements have resulted in a surging demand for applications with varying quality of service (QoS) requirements, including autonomous driving and virtual reality [1]. Although cloud computing can provide on-demand computing resources and services for tasks requiring high computational power or low latency, it has certain limitations [2]. A limitation is that cloud servers, located at the network core, are often far away from end users. When numerous users offload real-time tasks to the cloud center, heightened network congestion and increased security risks occur during data transmission [3]. Depending solely on a single cloud computing architecture is no longer adequate to meet the requirements of today's applications.

Mobile edge computing (MEC) has emerged as a promising computing paradigm, integrating mobile network edge communication with cloud computing services to offer computing services near mobile user devices (UDs) [4]. This feature enables mobile users to sequentially offload real-time tasks to MEC servers situated at the network's edge, thereby achieving low latency and high QoS without relying on cloud centers, a process known as computation offloading (CO) [5]. MEC serves as a complementary extension to cloud computing rather than a complete replacement. Nonetheless, traditional MEC heavily relies on fixedly deployed base station facilities, making it challenging to effectively respond to computing service requests in remote regions, disaster-affected areas, and during instances when facilities fail [6]. This limitation significantly restricts the applicability of traditional MEC.

* Corresponding author.
E-mail addresses: longxinzhang@hut.edu.cn (L. Zhang), runti_tan@163.com (R. Tan), yanfen.z@foxmail.com (Y. Zhang), jiwu_peng@hnu.edu.cn (J. Peng), luijing_cs@wust.edu.cn (J. Liu), lik@newpaltz.edu (K. Li).

https://doi.org/10.1016/j.sysarc.2024.103215
Received 11 March 2024; Received in revised form 2 June 2024; Accepted 18 June 2024
Available online 24 June 2024
1383-7621/© 2024 Published by Elsevier B.V.

Unmanned aerial vehicle (UAV) technologies have found extensive applications in public services, disaster relief, and fault detection due to their rapid advancements [7]. UAVs have emerged as a viable solution to overcome the above-mentioned challenges due to their flexible deployment, high mobility, and line-of-sight (LoS) communication abilities, giving rise to UAV-assisted MEC (UAV-MEC) [8]. In UAV-MEC, UAVs act as MEC servers, allowing them to adapt to unpredictable offloading environments and deliver ubiquitous computing services to users in a dynamic manner. The flexible deployment of UAVs to provide cellular services offers substantial improvements in cost-effectiveness and service efficiency compared with deploying additional MEC base stations. UAV-MEC establishes a distributed computing environment with mobile servers, where the servers (i.e., UAVs) are deployed in flight to provide flexible computing services [9].

However, UAVs encounter challenges in effectively providing comprehensive services to CO systems due to their limited computational power and battery life. These challenges pose significant obstacles to fully maximizing the aforementioned advantages of UAV-MEC. Meanwhile, numerous studies oversimplify tasks as independent entities, whereas real tasks often involve multiple subtasks with data dependencies, which can be represented as a directed acyclic graph (DAG) [10]. Additionally, the trajectory of UAVs is crucial in developing efficient CO strategies because UAVs deliver computational services based on their flight paths. Accordingly, the multi-device environment with dynamic UAV involvement introduces a significant number of high-dimensional state parameters. Traditional optimization algorithms typically require numerous iterations, posing challenges in adapting to real-time environmental changes and impeding the establishment of an optimal model [11].

Deep reinforcement learning (DRL)-based approaches can adaptively adjust CO policies in UAV-MEC by interacting with complex environments in real time compared with traditional methods [12]. DRL transforms the challenging sequential decision-making problem into a cumulative reward maximization problem within the framework of a Markov decision process (MDP), which has been regarded as a suitable method to search the asymptotically optimal solutions in time-varying edge environments [13]. DRL can iteratively learn and refine CO policies by utilizing a neural network, gradually converging toward the theoretical optimum. However, the hybrid actions performed by agents in UAV-MEC pose new challenges to the convergence speed and accuracy of DRL algorithms.

This study investigates the improved DRL-based UAV-assisted dependency-aware CO (UDCO) problem in collaborative computing systems to address the aforementioned challenges. First, we construct a UAV-assisted device–edge–cloud (D–E–C) collaborative computing system. D–E–C three-layer fusion integrates various computing resources to offer complementary computing services for dependent tasks. Second, we jointly optimize CO policies and UAV endurance while considering task-dependent and resource constraints. The optimization objective is to minimize the task completion cost (considering time delay and energy consumption) of all DAG tasks and UAV power consumption. Finally, we enhance the stability of the critic network self-training by optimizing the network update rule in actor–critic deep reinforcement learning (AC-DRL) and propose a maximum entropy RL (MERL)-based CO algorithm specifically designed for UAV-assisted dependency awareness scenarios.

The contributions of the study are as follows:

- It proposes a three-layer cooperative computing system involving UAV participation in the D–E–C framework. This system operates within a time-varying environment, accounting for channel transmission obstacles and stochastic user movements.
- It formulates a joint minimization problem that considers the task completion cost (TCC) and UAV energy consumption while accounting for resource and task-dependent constraints. A specialized MDP is designed to address this problem using DRL.
- It introduces a saturated training SAC-empowered UAV-assisted dependency-aware CO (STS-UDCO) algorithm to generate the optimal offloading strategy, where an adaptive saturation training rule (ASTR) is proposed to enhance the stability and learning capability in STS-UDCO. The experimental results reveal that STS-UDCO outperforms other advanced algorithms in terms of cost optimization. Additionally, STS-UDCO shows notable performance in terms of convergence and stability.

A shorter version of this work was accepted at the IEEE ICPADS conference in 2023 [14]. However, the conference version did not explore the complexities of task dependencies in practical scenarios. This extended version addresses this issue by introducing modifications to the MDP and utilizing a different optimization algorithm. Furthermore, this study enriches the discourse and comparison in the introduction and related work sections. This study provides more detailed structural diagrams and includes further analysis on the complexity of the developed algorithm. Additionally, a wider range of comparative algorithms are referenced to facilitate additional comprehensive experiments and a thorough evaluation of the algorithm performance.

The rest of the paper is organized as follows. Section 2 discusses the related work about CO schemes. Section 3 describes the dependency-aware CO system model for UAV-assisted D–E–C collaboration. Section 4 presents the problem formulation and MDP design under this model. Section 5 introduces the ASTR and STS-UDCO algorithm in detail. Section 6 evaluates the experimental performance of STS-UDCO and compares it with advanced algorithms. Finally, Section 7 concludes the paper.

2. Related work

In recent years, researchers have conducted extensive studies on the CO problem in MEC environments. This section introduces three types of existing CO schemes: CO schemes in conventional MEC, UAV-assisted CO solutions based on traditional optimization algorithms, and UAV-assisted CO solutions based on DRL algorithms. A summary and comparison of the related work are provided in Table 1.

2.1. CO schemes in conventional MEC

MEC brings significant improvements to user's QoS by leveraging its robust computing power and quick response time. Furthermore, MEC alleviates the service burden on centralized cloud centers and can collaborate with the cloud to deliver comprehensive and efficient computing services. Sahni et al. [15] conducted research on multi-hop partial offloading in collaborative MEC to minimize the average completion time. They demonstrated that this problem falls under the class of NP-hard problems. In their study, they devised a heuristic algorithm called joint partial offloading and flow scheduling (JPOFH), which uses the McCormick envelope to transform the problem into a linear programming problem. Sun et al. [16] proposed a cloud–edge computing-based method called joint vehicle task offloading and job scheduling (JVTR) to optimize the on-board task offloading and job scheduling process. JVTR minimizes time delay, energy consumption, and workload balance within the joint offloading environment of cloud, edge, and vehicles. Thereafter, they utilized an ant colony optimization (ACO) algorithm to generate an optimal offloading strategy. Wu et al. [17] designed a software-defined network-based MEC architecture (SDN-MEC) for data processing in the Industry 4.0 scenario. They developed a resource allocation algorithm leveraging stochastic game and prioritized experience replay (SGRA-PER). The experimental results confirmed the effectiveness and superiority of SDN-MEC and SGRA-PER. Ding et al. [18] classified the cooperative computing system into two architectural types, hierarchical (Hi) and horizontal (Ho), based on the visibility and accessibility of the cloud. They devised two CO algorithms based on non-cooperative game theory, termed as CO algorithm for hierarchical architectures (COAHx). Wu et al. [19] introduced

Table 1
Summary and comparison of recent work.

Algorithm	Technology	Optimization goals	CO type	UAV assistance	D–E–C fusion	Dependency
JPOFH [15]	Heuristic	Average time delay	Portion	×	✓	✓
JVTR [16]	Meta-heuristic	Energy consumption, latency, and workload balance	Binary	×	✓	×
SGRA-PER [17]	Game theory, DRL	Energy consumption and time delay	Binary	×	✓	×
COAHx [18]	Game theory	Energy consumption and time delay	Binary	×	✓	×
OPCO-LA [19]	DRL	Energy consumption and privacy volume	Binary	×	×	×
DRLCOSCM [20]	DRL	Service cost	Binary	×	✓	×
DQN-ETCM [21]	DRL, Meta-heuristic	Makespan	Binary	×	×	×
STMTO [22]	Heuristic	Energy consumption and time delay	Binary	✓	×	×
OPAD [23]	Heuristic	Computational speed	Portion	✓	×	×
DRLO [24]	Heuristic	Time delay	Binary	✓	✓	×
DDE [25]	Meta-heuristic	UAV energy consumption and time delay	Binary	✓	×	×
BO-MADDPG [26]	DRL, heuristic	Energy consumption	Binary	✓	×	×
EE-PPO [27]	DRL	Energy efficiency	Portion	✓	×	×
DQN-PER [28]	DRL	Energy consumption	Binary	✓	×	✓
DuelingDRL [29]	DRL	Energy consumption and time delay	Binary	✓	✓	×
SMOO [30]	DRL	Time delay	Portion	✓	×	×
STS-UDCO (Our work)	DRL	UAV energy Consumption and task completion cost	Binary	✓	✓	✓

an online privacy-aware IIoT computation offloading method based on Lyapunov optimization and actor–critic framework (OPCO-LA) to address the privacy and security concerns of IIoT users. The goal of this method is to mitigate privacy risks and lower system energy consumption. Zhou et al. [20] developed a DRL-based computation offloading and service caching mechanism (DRLCOSCM) for reducing the cost of cloud service centers while meeting delay constraints. They presented an algorithm using an asynchronous advantage actor–critic approach to handle the large-scale state space of DRLCOSCM. Zeng et al. [21] devised an edge task scheduling method based on an improved dual deep Q-network (DQN-ETCM) to optimize the maximum completion time. They utilized an improved particle swarm optimization algorithm for preprocessing to enhance training efficiency.

The aforementioned studies showcase innovative and remarkable CO schemes within MEC or D–E–C collaboration scenarios. They successfully address the optimization problem within the respective constructed frameworks, providing valuable insights for dealing with the CO problem within UAV-MEC environments.

2.2. *CO schemes in UAV-MEC using a traditional optimization algorithm*

In dynamic, hazardous, and resource-poor environments, the collaboration between UAVs and traditional computing systems becomes crucial in expanding service capabilities and improving QoS. Guo et al. [22] proposed the smart trusted multi-UAV task offloading (STMTO) approach to minimize system energy consumption and computational delay. They devised a two-way auction mechanism for multi-to-multi task offloading, along with a server trust evaluation method to maximize device utility. He et al. [23] introduced two real-time distributed joint optimal policies to address the optimal power allocation and deployment (OPAD) problem. Their approach enables efficient over-the-air multi-hop edge computing with a multi-UAV relay network. Chen et al. [24] conducted a study on CO in the air–heaven–earth computing network. They proposed a distributionally robust latency optimization (DRLO) algorithm to minimize task delays while respecting energy consumption constraints. Mousa et al. [25] developed a discrete differential evolution (DDE) algorithm with a new variant and crossover operator to determine cluster specifications and optimal UAV cluster service paths by using an ACO approach. The goal of DDE is to address the limitations of UAV capability in MEC environments and outdated benchmark meta-heuristic algorithms.

Despite the significant advancements in the field of CO research under UAV-MEC using traditional optimization algorithms, these methods encounter challenges in terms of efficiently handling complex environment models or coping with rapid and massive variations in parameters.

2.3. *CO schemes in UAV-MEC using the DRL algorithm*

DRL combines the strengths of RL and deep neural networks, allowing for the accurate identification of state differences through the powerful perception and representation capabilities of neural networks. This algorithm explores the environment under the MDP model and leverages accumulated experience to effectively navigate the solution space of the problem. Gong et al. [26] developed a MADDPG-based multi-UAV trajectory planning algorithm to address the task offloading problem in a multi-UAV multi-hop network. They utilized Bayesian optimization (BO) to estimate UAV actions and enhance learning efficiency. However, they did not account for inter-task dependencies in multiple hops. Li et al. [27] studied the optimization problem of wireless power supply and energy efficiency (EE) maximization in UAV-MEC computing networks and introduced an EE-maximization proximal policy optimization (EE-PPO) algorithm. However, the one-time sampling approach of EE-PPO could be wasteful and inefficient in resource utilization and historical experience. Wei et al. [28] developed a task offloading algorithm called deep Q-network with prioritized experience replay (DQN-PER). The algorithm addresses the joint optimization problem of UAV formation deployment, UAV trajectory optimization, and task-dependent scheduling in UAV-MEC environments. However, considering the limitations of DQN as a value-based algorithm for handling continuous actions [31], the paper discretized the flight angles into eight directions, which affected the models accuracy. Jiang et al. [29] introduced an efficient task offloading algorithm based on dueling DQN (DuelingDRL) but did not consider the computing power of devices and cloud centers. Peng et al. [30] proposed a multi-objective optimization algorithm using SAC for service caching, task offloading, and resource allocation (SMOO). However, the original SAC algorithm used in their work encounters challenges in critic fitting efficiency.

In summary, existing research has explored the CO problem in UAV-MEC environments. However, they have overlooked the potential efficiency of collaborative computation in UAV-MEC, the realistic task dependencies, and the comprehensiveness of the CO environment. A heterogeneous system can effectively reduce costs and enhance system reliability [32]. Moreover, these studies have neglected the large-scale changes in complex computing environments, which pose challenges for the convergence of CO algorithms. This study focuses on the dependency-aware CO problem within a UAV-enabled D–E–C cooperative system to address these issues. The proposed solution, STS-UDCO, aims to overcome challenges related to convergence speed and accuracy in the aforementioned complex environment.

3. UAV-assisted D–E–C cooperative dependency-aware CO system

This section initially presents the overview model of the UAV-assisted dependency-aware CO system in D–E–C collaborative computing. Subsequently, the system is elaborated in three parts: mobility,

Table 2
Description of notations.

Notation	Definition
UD_i	ith UD
$DAG(j)$	DAG structure of the jth application
$Task(j)$	Subtask set of the jth application
$Con(j)$	Set of dependency relationships for the jth application
$\mathrm{con}_j(\mathrm{task}_{j,l}, \mathrm{task}_{j,n})$	Dependency of the lth task and the nth task in group j
$l_{j,n}$	Transmission amount of $\mathrm{task}_{j,n}$
$d_{j,n}$	Calculation data of $\mathrm{task}_{j,n}$
$t^s_{j,n}$	Start time of $\mathrm{task}_{j,n}$
$t^e_{j,n}$	End time of $\mathrm{task}_{j,n}$
$\mathcal{I}$, I	Set of UDs and number of UDs
$\mathcal{M}$, M	Set of computing facility types and number of computing facility types
$\mathcal{N}$, N	Set of tasks and number of tasks
$\mathcal{T}$, T	Set of time slots and length of time sequence
L_i, L_u, L_c	Location of UD_i, UAV, and cloud server
$o_u(t)$, $o_i(t)$	Flight angle of UAV and UD_i
$v_u(t)$, $v_i(t)$	Flight speed of UAV and UD_i
$E_f(t)$, $E_h(t)$	Flight and hovering energy consumption of UAV
$T^{cal}_i(j,n)$	Calculation time delay on UD_i
$E^{cal}_i(j,n)$	Calculation energy consumption on UD_i
$T^{tra}_{i,u}(j,n)$	Transmission latency between UD_i and UAV
$T^{cal}_u(j,n)$	Calculation time delay on UAV
$E^{tra}_{i,u}(j,n)$	Transmission energy consumption between UD_i and UAV
$E^{cal}_u(j,n)$	Calculation energy consumption on UAV
$T^{tra}_{u,c}(j,n)$	Transmission latency between the UAV and the cloud center
$T^{cal}_c(j,n)$	Calculation time delay on cloud center
$E^{tra}_{u,c}(j,n)$	Transmission energy consumption between the UAV and the cloud center
$E^{cal}_c(j,n)$	Calculation energy consumption on cloud center
$h_{i,u}$, $h_{u,c}$	Channel gain from UD_i to UAV and from UAV to cloud center
$r_{i,u}$, $r_{u,c}$	Transmission rate from UD_i to UAV and from UAV to cloud
h_0	Channel gain per unit distance
$p_{i,u}(t)$	Judgment factor for the existence of NLoS communication between UD_i and UAV
W_{NLoS}	Channel transmission loss caused by NLoS
P^{up}_i, P^{up}_u	Transmission power of UD_i and UAV
f_i, f_u, f_c	CPU calculation frequency of UD_i, UAV, and cloud center

communication, and CO model. Table 2 displays the primary notations utilized in this work to enhance clarity.

3.1. System model

Fig. 1 illustrates the system model of the UAV-assisted D–E–C three-layer cooperative dependency-aware CO framework. This figure depicts a UAV operating within a square region and equipped with a micro-MEC server to manage DAG flow tasks with dependencies related to UDs. The environment is supported by three computing facilities denoted by m, each offering computing services with varying capabilities. These facilities range from weak to strong, including the UD ($m = 0$), the UAV ($m = 1$), and a cloud server situated at a considerable distance from the service area ($m = 2$). The cloud server consistently possesses superior computational and energy resources compared with the UAV.

In this scenario, the entire time sequence $\mathcal{T}$ is divided into T time slots. At the end of each time slot $t \in \mathcal{T} = \{1, 2, \dots, T\}$, the UDs in the dynamic environment randomly move in small increments and assess whether obstacles hinder communication with the UAV. This assessment determines whether the link between them is LoS or a non-LoS (NLoS). Subsequently, the UAV determines whether to offer computing or transmission services to a specific UD based on the CO policy. If either the UAV or UD can process the task efficiently, then the task is handled directly by the UAV or UD. Otherwise, the task is forwarded to the cloud center for computation. Therefore, the CO model comprises four phases: (1) UD to UAV, (2) computation at the UAV, (3) UAV to cloud, and (4) computation at the cloud. Given that the small volume of the computational task result and the ample downlink between the UAV and the cloud center, this study assumes a negligible overhead in the backhaul for transmitting the task result.

In practical scenarios, tasks often demonstrate dependencies. We consider an environment consisting of J groups of data-dependent applications. Each group encompasses a common set of subtasks with sequential relationships, autonomously produced by multiple devices dispersed across the environment. A widely adopted method for representing dependent tasks is the utilization of DAGs, which constrain the sequential execution order of tasks [33]. Therefore, the application of DAG in the environment can be defined as follows:

$$DAG(j) = \left\{Task(j), Con(j) \mid j \in [1, J]\right\}, \tag{1}$$

where $Task(j) = \{\mathrm{task}_{j,1}, \mathrm{task}_{j,2}, \dots, \mathrm{task}_{j,n}\}$ denotes the set of subtasks of the j-th application, and $n \in \mathcal{N} = \{1, 2, \dots, N\}$. $Con(j)$ represents the set of task dependency constraints for the jth application. $\mathrm{con}_j(\mathrm{task}_{j,l}, \mathrm{task}_{j,n})$ denotes the dependency constraint between $\mathrm{task}_{j,l}$ and $\mathrm{task}_{j,n}$. This notion indicates that $\mathrm{task}_{j,n}$ must be executed after $\mathrm{task}_{j,l}$ is completed. When $\mathrm{task}_{j,n}$ does not have any direct predecessor task, $\mathrm{task}_{j,n}$ represents the start task of $DAG(j)$. If $\mathrm{task}_{j,n}$ has no direct successor task, then $\mathrm{task}_{j,n}$ represents the end task in $DAG(j)$. As shown in Fig. 1, $\mathrm{task}_{j,1}$ and $\mathrm{task}_{j,6}$ denote the start and end tasks in this application, respectively.

Each task in applications is defined as $\mathrm{task}_{j,n} = \{\mathrm{UD}_i, l_{j,n}, d_{j,n}, t^s_{j,n}, t^e_{j,n}\}$, where $l_{j,n}$ represents the transmission amount of $\mathrm{task}_{j,n}$ (in bits), and $d_{j,n}$ denotes the calculation data of $\mathrm{task}_{j,n}$ (in number of CPU cycles per bit). UD_i indicates the generator of $\mathrm{task}_{j,n}$ is the ith UD, where $i \in \mathcal{I} = \{1, 2, \dots, I\}$. $t^s_{j,n}$ and $t^e_{j,n}$ represent the start time and end time of $\mathrm{task}_{j,n}$, respectively.

3.2. Mobility model

Assuming that the UAV is flying at a fixed altitude H to simplify the model, the coordinates of a 3D Cartesian coordinate system can be expressed as $L_u(t) = \left[x_u(t), y_u(t), \mathrm{H}\right]$. The coordinates of UD_i can be denoted as $L_i(t) = \left[x_i(t), y_i(t), 0\right]$. Let Δ be the duration of time slot t. The UAV receives a flight command at time slot t, which includes the flight angle $o_u(t)$ and flight speed $v_u(t)$. The coordinates of the UAV in time slot $t+1$ are updated by the following equations (Fig. 2):

$$x_u(t+1) = x_u(t) + \cos\left(o_u(t)\right) \times D_u(t), \tag{2}$$

$$y_u(t+1) = y_u(t) + \sin\left(o_u(t)\right) \times D_u(t), \tag{3}$$

where $D_u(t) = v_u(t) \times \Delta$ represents the flight distance of UAV in time slot t.

Considering that UD_i executes a brief random movement $D_i(t)$ within the environment, the position of UD_i in time slot $t+1$ is obtained as follows:

$$L_i(t+1) = \left[x_i(t) + \cos\left(o_i(t)\right) \times D_i(t), y_i(t) + \sin\left(o_i(t)\right) \times D_i(t), 0\right]. \tag{4}$$

The constraints of the UAV and UD_i within a service range of length L and width W are expressed as follows:

$$L_u(t) \in \left\{x_u(t), y_u(t) \mid x_u(t) \in [0, L], y_u(t) \in [0, W]\right\}, \forall t, \tag{5}$$

$$L_i(t) \in \left\{x_i(t), y_i(t) \mid x_i(t) \in [0, L], y_i(t) \in [0, W]\right\}, \forall i, t. \tag{6}$$

Moreover, the movement angle and speed must comply with the following constraints to ensure the legitimate movement of the UAV and UD_i:

$$0 \le o_u(t) \le 2\pi, \forall t, \tag{7}$$

$$0 \le o_i(t) \le 2\pi, \forall t, \tag{8}$$

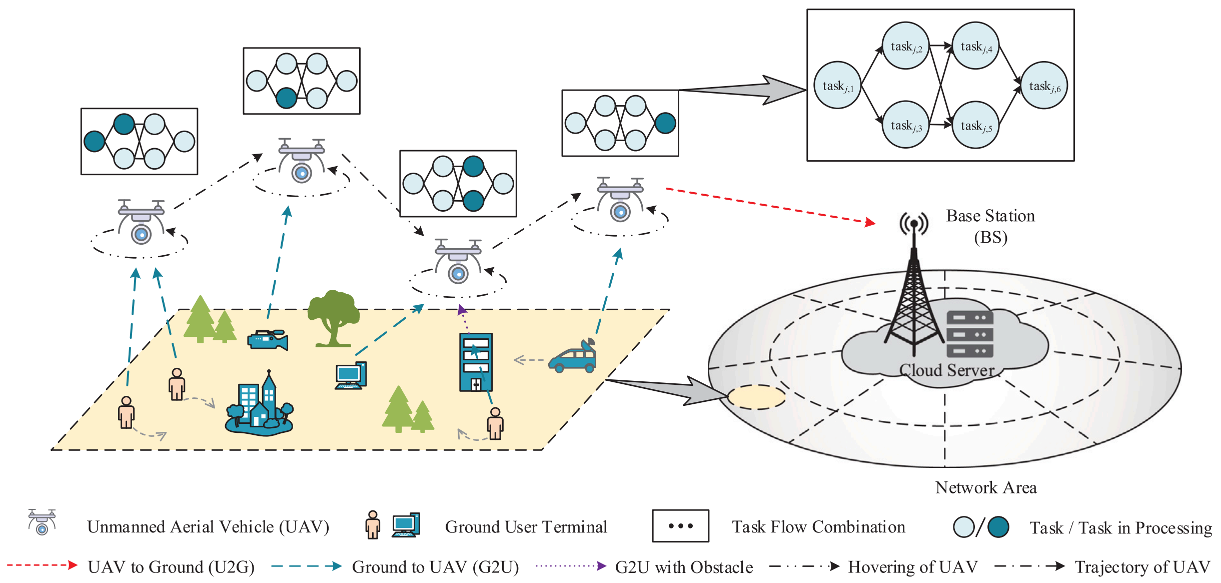

Fig. 1. UAV-assisted D–E–C cooperative dependency-aware CO system.

$$0 \le v_u(t) \le v_{\max},\ \forall t, \tag{9}$$

where $v_{\max}$ is the maximum speed of the UAV. The flight energy consumption of the UAV exhibits an approximate linear relationship with its weight [34]. Therefore, the flight energy consumption $E_f(t)$ at time slot t is defined as follows:

$$E_f(t) = v_u^2(t) \times q_u \times D_u(t) \times 0.5, \tag{10}$$

where q_u denotes the mass of the UAV. According to Ref. [35], the hover energy consumption $E_h(t)$ is estimated by

$$E_h(t) = n_r \sqrt{\frac{(q_u g)^3}{2\rho\pi\psi^2}}, \tag{11}$$

where n_r is the number of rotating wings, g is the universal gravitational constant, ρ is the fluid density of air, and ψ is the radius of the rotating wings.

3.3. Communication model

In this system, UD$_i$ establishes LoS communication with the UAV through a wireless uplink transmission link. The mobility model provides the 3D coordinates of the UAV and UD$_i$. Subsequently, the 3D Euclidean distance can be expressed as $D_{i,u}(t) = \sqrt{\left(L_u(t) - L_i(t)\right)^2 + \mathrm{H}^2}$.

The maximum communication distance of the UAV is set to $R_{\max}$ to ensure that the UAV can provide services within a restricted communication range. The aforementioned approach enables UD$_i$ to maintain an effective connection with the UAV. This constraint is expressed as follows:

$$D_{i,u}(t) \le R_{\max},\ \forall i \in \mathrm{I}. \tag{12}$$

The channel gain with the UAV and UD$_i$ at time slot t is defined as follows, similar to Ref. [36]:

$$h_{i,u}(t) = h_0 D_{i,u}^{-2}(t) = \frac{h_0}{\left(L_u(t) - L_i(t)\right)^2 + \mathrm{H}^2}, \tag{13}$$

where h_0 denotes the channel gain per unit distance. Obstacles give rise to NLoS links during communication in real-world environments [37].

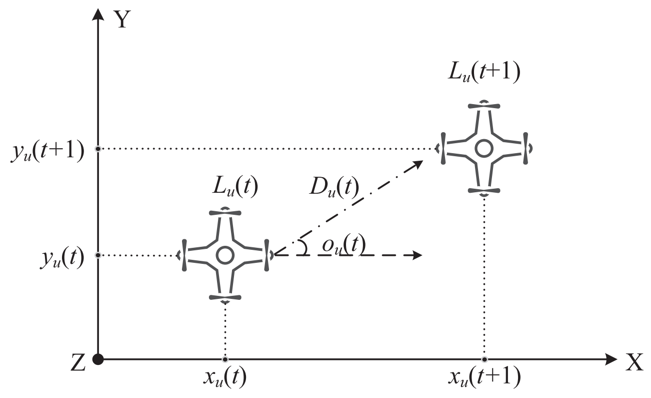

Fig. 2. UAV movement.

Consequently, the wireless channel transmission rate between UD$_i$ and UAV, considering the NLoS channel, can be calculated by

$$r_{i,u}(t) = B\log_2\left(1 + \frac{P_i^{up} h_{i,u}(t)}{\sigma_{NLoS}^2 + p_{i,u}(t) W_{NLoS}}\right), \tag{14}$$

where B represents the channel bandwidth, P_i^{up} denotes the uplink transmit power of UD$_i$, σ_{NLoS}^2 represents the Gaussian white noise associated with obstacles, $p_{i,u}$ indicates the occurrence of an NLoS communication link between the UAV and the UD$_i$ during time slot t, and W_{NLoS} represents the transmission loss due to the NLoS channel. The channel gain between the UAV and the cloud center at time slot t is also defined as follows:

$$h_{u,c}(t) = \frac{h_0}{\left(L_c(t) - L_u(t)\right)^2 + \mathrm{H}^2}, \tag{15}$$

where $L_c(t)$ denotes the coordinates of the cloud center. Given that the UAV and the cloud center have superior LOS communication conditions, the wireless channel transmission rate between them is calculated as follows:

$$r_{u,c}(t) = B\log_2\left(1 + \frac{P_u^{up} h_{u,c}(t)}{\sigma^2}\right), \tag{16}$$

where P_u^{up} represents the uplink transmit power of the UAV, and σ^2 denotes the power of Gaussian white noise.

3.4. CO model

The CO evaluation indexes in this study include time delay and energy consumption, which encompass task calculation and task uplink transmission across various computing facilities.

3.4.1. Local computing model

The time delay and energy consumption of $\text{task}_{j,n}$ executed by UD_i are solely dependent on the computational power and loss of UD_i. The local time delay $T_i(j,n)$ is expressed as follows, consistent with [38]:

$$T_i(j,n) = \frac{sd_{j,n}}{f_i}, \tag{17}$$

where f_i represents the CPU computing frequency of UD_i, and s indicates the number of CPU cycles required to process one unit bit of data. The local energy consumption $E_i(j,n)$ is formulated as follows, similar to [38]:

$$E_i(j,n) = P_i \times T_i(j,n) = \kappa f_i^2 sd_{j,n}, \tag{18}$$

where $P_i = \kappa f_i^3$ represents the computing power of UD_i [39], and κ is a constant associated with the chip [40].

3.4.2. UAV edge computing model

When $\text{task}_{j,n}$ is transferred to the UAV, the time delay of $\text{task}_{j,n}$ typically includes the uplink transmission time delay from UD_i to the UAV and the computing time delay of the UAV. The edge time delay $T_{i,u}(j,n)$ is denoted as follows:

$$T_{i,u}(j,n) = T_{i,u}^{tra}(j,n) + T_u^{cal}(j,n) = \frac{l_{j,n}}{r_{i,u}(t)} + \frac{sd_{j,n}}{f_u}, \tag{19}$$

where f_u denotes the CPU computing frequency of the UAV.

The edge energy consumption $E_{i,u}(j,n)$ also considers the uplink transmission energy consumption from UD_i to the UAV and the computation energy consumption of the UAV should be considered in this model. Specifically, the transmission energy consumption $E_{i,u}^{tra}(j,n)$ is denoted as $E_{i,u}^{tra}(j,n) = P_i^{up} \times T_{i,u}^{tra}(j,n) = P_i^{up} l_{j,n}/r_{i,u}(t)$, and the computational energy consumption $E_u^{cal}(j,n)$ is denoted as $E_u^{cal}(j,n) = P_u \times T_u^{cal}(j,n) = \kappa f_u^2 sd_{j,n}$. Therefore, $E_{i,u}(j,n)$ is calculated as follows:

$$E_{i,u}(j,n) = E_{i,u}^{tra}(j,n) + E_u^{cal}(j,n) = \frac{P_i^{up} l_{j,n}}{r_{i,u}(t)} + \kappa f_u^2 sd_{j,n}, \tag{20}$$

where $P_u = \kappa f_u^3$ represents the computing power of the UAV.

3.4.3. Cloud computing model

If $\text{task}_{j,n}$ is chosen to be further offloaded to the cloud center for execution, then the total cost (comprising time delay and energy consumption) of $\text{task}_{j,n}$ can be divided into three parts: the transmission cost before offloading to the UAV, the transmission cost from the UAV to the cloud center, and the computation cost at the cloud center. The cloud computing delay $T_{i,c}(j,n)$ is denoted as follows:

$$T_{i,c}(j,n) = T_{i,u}^{tra}(j,n) + T_{u,c}^{tra}(j,n) + T_c^{cal}(j,n) = \frac{l_{j,n}}{r_{i,u}(t)} + \frac{l_{j,n}}{r_{u,c}(t)} + \frac{sd_{j,n}}{f_c}, \tag{21}$$

where f_c denotes the CPU computing frequency of the cloud server.

The aforementioned three-stage time delay in the cloud computing model corresponds to three-stage energy consumption. Specifically, the uplink transmission energy consumption from the UAV to the cloud center is represented as $E_{u,c}^{tra}(j,n) = P_u^{up} \times T_{u,c}^{tra}(j,n) = P_u^{up} l_{j,n}/r_{u,c}(t)$, while the computing energy consumption of the cloud center is denoted as $E_c^{cal}(j,n) = P_c \times T_c^{cal}(j,n) = \kappa f_c^2 sd_{j,n}$. Thereafter, the cloud energy consumption $E_{i,c}(j,n)$ can be expressed as follows:

$$E_{i,c}(j,n) = E_{i,u}^{tra}(j,n) + E_{u,c}^{tra}(j,n) + E_c^{cal}(j,n) = \frac{P_i^{up} l_{j,n}}{r_{i,u}(t)} + \frac{P_u^{up} l_{j,n}}{r_{u,c}(t)} + \kappa f_c^2 sd_{j,n}, \tag{22}$$

where $P_c = \kappa f_c^3$ represents the computing power of the cloud center.

4. Problem formulation and MDP design

This section formulates a joint optimization problem for the CO policy and UAV endurance under task-dependent and resource constraints within the aforementioned model. This problem is transformed into an MDP specifically tailored to handle high-dimensional states and hybrid actions and efficiently apply the DRL approach.

4.1. Problem formulation

Given the system model, the computational completion cost of tasks is introduced as the primary evaluation metric of the CO strategy in this study.

Definition 1 (*Task Completion Cost (TCC)*)**.** TCC is defined as the weighted sum of time delay and energy consumption associated with the computation and transmission of $\text{task}_{j,n}$ related to UD_i.

Thus, the TCC of $\text{task}_{j,n}$ can be represented differently based on three distinct offloading goals:

$$\text{TCC}_{j,n} = \begin{cases} \delta \times T_i(j,n) + \theta \times E_i(j,n), & \text{if } \gamma_0(j,n) = 1; \\ \delta \times T_{i,u}(j,n) + \theta \times E_{i,u}(j,n), & \text{if } \gamma_1(j,n) = 1; \\ \delta \times T_{i,c}(j,n) + \theta \times E_{i,c}(j,n), & \text{otherwise}, \end{cases} \tag{23}$$

where $\gamma_m(j,n) \in \{0,1\}$ is a binary variable, indicates that $\text{task}_{j,n}$ can only be executed by the mth computing device. $m \in \mathcal{M} = \{0,1,2\}$ represents three types of computing facilities: user terminals, UAV onboard edge servers, and cloud centers. When $\gamma_1(j,n) = 1$, $\text{task}_{j,n}$ is divided into an edge offloading state, which will be offloaded until the UAV completes the computing task and then transmitted back to the corresponding user. At this point, $\gamma_0(j,n) = 1$ and $\gamma_2(j,n) = 1$ are set to 0. δ and θ represent the time delay and energy consumption weighting factors, respectively. The current TCC of the task is calculated by combining the two weighted factors.

The above-mentioned CO environment considers the dependencies between tasks, implying that when a task is executed, all of its direct predecessors must have already been completed. Let $\text{task}_{j,l}$ denote all direct predecessors that match $\text{con}_j(\text{task}_{j,l}, \text{task}_{j,n})$.

Definition 2 (*Task Time Dependency (TTD)*)**.** TTD is defined as the condition that the start time $t_{j,n}^s$ of $\text{task}_{j,n}$ must be greater than or equal to the end time $t_{j,l}^e$ of all of its direct predecessors' $\text{task}_{j,l}$. $t_{j,l}^s$ and $t_{j,n}^e$ together form a set of dependent time node pairs, and $\text{task}_{j,l}$ and $\text{task}_{j,n}$ must satisfy $\text{con}_j(\text{task}_{j,l}, \text{task}_{j,n})$. Therefore, the TTD is calculated as follows:

$$t_{j,l}^e \le t_{j,n}^s, \forall \text{con}_j(\text{task}_{j,l}, \text{task}_{j,n}). \tag{24}$$

$t_{j,n}^e$ is represented as follows in accordance with the services provided by various computing facilities:

$$t_{j,n}^e = \begin{cases} t_{j,n}^s + T_i(j,n), & \gamma_0(j,n) = 1; \\ t_{j,n}^s + T_{i,u}(j,n), & \gamma_1(j,n) = 1; \\ t_{j,n}^s + T_{i,c}(j,n), & \gamma_2(j,n) = 1. \end{cases} \tag{25}$$

Given that $t_{j,n}^s$ relies on the maximum end time of its direct predecessors, it is expressed as $t_{j,n}^s = \max\{t_{j,l}^e\}$, $\forall \text{con}_j(\text{task}_{j,l}, \text{task}_{j,n})$.

This study aims to simultaneously minimize the TCC of the entire CO process and the energy consumption of the UAV for all UD_i while taking into account task-dependent and resource constraints. The optimization problem is formulated as follows:

$$
\begin{aligned}
(P_1):\quad & \underset{\{\delta,\theta,o,v\}}{\text{minimize}} \left\{ \sum_{t=1}^{T}\sum_{j=1}^{J}\sum_{n=1}^{N}\sum_{m=0}^{2} \gamma_m(j,n)\left(\mathrm{TCC}_{j,n} + E_f(t) + E_h(t)\right)\right\},\\
\text{s.t.}\quad & C_1: \delta+\theta=1,\\
& C_2: \gamma_m(j,n)\in\{0,1\},\forall m,j,n,\\
& C_3: \sum_{m=0}^{2}\gamma_m(j,n)=1,\forall j,n,\\
& C_4: \sum_{t=1}^{T}\left(E_f(t)+E_h(t)+E_s(t)\right)\le E_b,\forall t,\\
& C_5: \text{Eq. (5)–Eq. (9), Eq. (12)},\\
& C_6: \text{Eq. (24)}.
\end{aligned}
\tag{26}
$$

The above-mentioned constraints are explained as follows.

Constraint C_1 represents the weight constraint, indicating that the sum of the ratio of delay to energy consumption cannot exceed one. Constraints C_2 and C_3 illustrate the task attribution constraints, specifying that $\mathrm{task}_{j,n}$ can select only one of the local, UAV, or cloud center computing facilities to handle its tasks. Constraint C_4 denotes the UAV power constraint, where the service energy consumption $E_s(t)$ consists of $E_u^{cal}(j,n)$ and $E_u^{tra}(j,n)$, while E_b represents the total battery life. Constraint C_5 specifies the legal action constraints, encompassing that the motion of the UAV cannot exceed the legal environmental boundaries, angle range, and maximum communication distance. Constraint C_6 outlines the task dependency constraint, showing that any sub task's service must start after all its parent tasks are completed.

Given that $\gamma_m(j,n)$ is subject to an integer constraint, it becomes a critical factor in solving P_1. Consequently, P_1 transforms into a mixed integer nonlinear programming (MINLP) problem, which is typically NP-hard and non-convex. Additionally, the interplay and conflict between energy consumption and time delay impede the joint optimization of these two factors. These challenges greatly influence the convergence speed and accuracy of algorithms. Therefore, we convert P_1 into an MDP suitable for RL processing, aiming to propose a MERL-based CO algorithm to overcome the complexity of P_1.

4.2. MDP design under high-dimensional states

The system cost is influenced by the observation state of the environment and the UAV action. When the current action is applied to the current state, it triggers the environment to transition into the next state. At this point, P_1 can be structured as an MDP consisting of three key elements: observation space, action space, and reward function.

4.2.1. Observation space

In the environment described in this study, the MDP observation space comprises the observed states of the UAV, the UDs, and the current environment. The observation space is modeled as

$$S=\left\{s_t|s_t=\left\{E_r(t), L_u(t), d_r(t), L_i(t), d_i(t), Con(t), p_{i,u}(t)\right\}\right\}, \tag{27}$$

where s_t is the current environmental observation at time slot t, $E_r(t)$ denotes the remaining power of the UAV at time slot t, $d_r(t)$ represents the remaining tasks in the environment at time slot t, $d_i(t)$ signifies the tasks waiting to be offloaded by UD_i at time slot t, and $Con(t)$ indicates the set of dependencies existing at time slot t.

We also apply normalization operations to the entire observation space to enhance the processing efficiency of DRL approach on input observations. This normalization enables the neural network to adjust to varying input variables.

Definition 3 (*Observation Normalization (OBN)*). OBN utilizes the difference between the maximum $item_{\max}$ and the minimum $item_{\min}$ of each observation state $item_t$ as a normalization factor to normalize $item_t$. The OBN for each element in S is defined as follows:

$$OBN(item_t)=item_t/(item_{\max}-item_{\min}),\forall item_t\in s_t, \tag{28}$$

where $item_t$ represents the seven state elements from $E_r(t)$ to $p_{i,u}(t)$ in the observation space.

4.2.2. Action space

The UAV takes actions based on the current state of the environment, which includes the target user, motion parameters, and selected server. The actions in this environment are hybrid, consisting of discrete and continuous action spaces. Discrete action spaces consist of service targets UD_i and server targets s, while continuous action spaces encompass the flight parameters $o_u(t)$ and $v_u(t)$ of the UAV. Therefore, the action space is modeled as follows:

$$A=\left\{a_t|a_t=\left\{\mathrm{UD}_i, o_u(t), v_u(t), \gamma_m(j,n)\right\}\right\}, \tag{29}$$

where a_t is the action taken at time slot t. Continuous action spaces pose significant challenges for solving problems using DRL algorithms because methods, such as DQN require discretization operations. Unlike value-based DRL approaches that produce probability distributions of actions, the SAC algorithm leveraged in this study utilizes Gaussian distributions to model the mean and covariance of actions, providing action samples. This mechanism is better suited for handling continuous actions in the UAV-MEC model.

A mapping pair between continuous and discrete actions is established to achieve SAC output for discrete actions. Specifically, discrete actions are encoded as continuous actions based on mapping relationships, and their ownership within segmented continuous space is then allocated to achieve a bidirectional mapping with discrete actions.

4.2.3. Reward function

After taking actions, immediate feedback is crucial to evaluate the effectiveness of the decision. A well-designed reward can effectively reflect the feedback from the current environment, guiding agents to learn and accelerate convergence. The reward function is modeled as follows:

$$R=r(s_t,a_t)=-\left(\mathrm{TCC}_{j,n}+E_f(t)+E_h(t)\right), \tag{30}$$

where $r(s_t,a_t)$ denotes the immediate reward that can be obtained by taking action a_t in state s_t. Given that P_1 aims to jointly optimize the TCC and UAV energy consumption, we set the reward value based on the minimization extent of this optimization objective. This step ensures that the reward aligns with the principles of agent training by maximizing the negative value of the objective.

5. STS-UDCO algorithm

Building upon the previous model and formulation, this section highlights the advantages and limitations of the MERL and AC frameworks. This mechanism will serve as a foundation for introducing an ASTR and elucidating the process of the STS-UDCO algorithm.

5.1. MERL and SAC under UAV-assisted DAG-CO

In traditional RL frameworks, the commonly discussed problem revolves around how an agent can maximize its gains in a complex environment [41]. Unlike traditional RL, the primary goal of MERL is to maximize the entropy of the strategy while simultaneously pursuing its benefits. This approach encourages agents to take actions in a more random manner. MERL is well suited for dealing with the complex strategy pursuit involved in D–E–C collaborative computing with UAV participation and dependent task flow. In this context, the optimal strategy learning is denoted as follows:

$$\pi^* = \arg\max_{\pi^*} \mathbb{E}_{(s_t, a_t)\sim\rho_\pi} \left[\sum_{t=0}^{T} r(s_t, a_t) + \varepsilon H\left(\pi(\cdot|s_t)\right) \right], \tag{31}$$

where ε represents the temperature coefficient, responsible for controlling the degree of randomness in the strategy. $H(\pi(\cdot|s_t)) = \mathbb{E}_{a\sim\pi(\cdot|s_t)}\left[-\log\pi(a|s_t)\right]$ denotes the information entropy, which utilizes logarithmic probability to quantify the specific measure of randomness in strategy π within s_t. The emphasis on strategy entropy empowers MERL with robust exploration capabilities and stability, allowing it to avoid local optima and capture multiple extreme points.

The SAC algorithm serves as an AC architecture algorithm within DRL and MERL frameworks, featuring two types of networks: actor network and critic network.

(1) **Actor**. The actor network interacts with the environment and generates policy actions following a Gaussian distribution in a deterministic policy.

(2) **Critic**. The critic network learns a value function from the experiences generated by the interactions of the actor network. This network uses this function to assess the subsequent actions and facilitate the optimization of the actor network parameters.

Furthermore, SAC introduces adaptive entropy adjustment, which dynamically adjusts the entropy level based on the fitting condition. This mechanism allows for flexible control of the stochastic exploration intensity of the strategy. The interplay and co-optimization of the actor and critic networks contribute to convergence efficiency and stability during agent training. However, the actor network relies on a well-trained critic network to provide accurate gradients for optimization to achieve the aforementioned advantages.

5.2. ASTR-based AC optimization scheme

In the AC architecture, the actor and critic networks mutually optimize and evolve together. However, frequent changes in action samples generated by the actor network can result in underfitting of the critic network, affecting the overall training stability and introducing gradient noise. Drawing from insights and applications of generative adversarial networks [42], the critic network can achieve a highly accurate fitting ability through more extensive training. This mechanism involves applying a more saturated training frequency to the critic compared to the actor within the AC structure. With a well-trained critic, this mechanism can better guide the actor network in developing appropriate strategies.

Therefore, we propose an ASTR designed to dynamically adjust the update rates of the actor and critic using two main strategies. First, we dynamically utilize a higher learning rate (LR) for the critic compared with the actor through the Adam optimizer. Second, our approach prioritizes multiple updates to the critic before updating the actor. Consequently, ASTR functions as an adaptive updating algorithm that dynamically establishes the update frequency of the critic based on the current fitting status of the critic network. ASTR determines the updating threshold of the critic through τ, which is expressed as follows:

$$\tau = \exp\left(-\chi^2\right), \tag{32}$$

where $\chi = 0.995 \times \chi + 0.005 \times F_Q(\beta)$ denotes the critic saturation factor, starting at one and updated following each iteration of the critic loss function $F_Q(\beta)$. ASTR continuously refreshes all critic networks, maintaining the critic update frequency beneath a preset upper threshold $b_{\max}$. The actor update occurs after meeting a judgment condition, defined as $a/b < 1/(2-\tau)$, $b \in \{1, 2, \dots, b_{\max}\}$, where a is a scalar manually adjusted to control the minimum update frequency of the critic. This method guarantees the critic's adequate training relative to the current fitting condition of $F_Q(\beta)$, thus reducing the adverse effects of its abnormal fitting on overall stability. Algorithm 1 elaborates the detail of ASTR.

In ASTR, the network parameters of the actor and critic are alternately updated using soft policy improvement and soft policy evaluation, integrating the adaptive update mechanism [39].

5.2.1. Soft policy evaluation

The critic maintains two sets of main and target networks to mitigate the overestimation issue during value evaluation. The parameters of the main critic network β are updated using the mean square error (MSE) loss function. The Bellman equation for $Q^*_\pi(s_t, a_t)$ is estimated as follows, incorporating the strategy entropy value, as per Eq. (31):

$$Q^*_\pi(s_t, a_t) = r(s_t, a_t) + \gamma \mathbb{E}_{(s_{t+1}, a_{t+1})}\left[Q^*(s_{t+1}, a_{t+1}) - \varepsilon \log\left(\pi(a_{t+1} \mid s_{t+1})\right)\right], \tag{33}$$

where γ denotes the reward discount factor. At this stage, the update function of β can be expressed by minimizing the soft Bellman residuals [43], which are calculated as follows:

$$F_Q(\beta) = \mathbb{E}_{(s_t, a_t, s_{t+1})\sim D}\left[\frac{1}{2}\left(Q_\beta(s_t, a_t) - \hat{Q}(s_t, a_t)\right)^2\right], \tag{34}$$

where $\hat{Q}(s_t, a_t) = r(s_t, a_t) + \gamma\left(Q_{\hat{\beta}}(s_{t+1}, a_{t+1}) - \varepsilon \log(\pi(a_{t+1}|s_{t+1}))\right)$ represents the learning target of the critic network, comprising $r(s_t, a_t)$ and the future reward. $\hat{\beta}$ denotes the target network parameter. $Q_\beta(s, a)$ and $Q_{\hat{\beta}}(s, a)$ represent the state–action value functions of the main critic network and its corresponding target network.

The target critic network updates its parameters through soft updates to maintain stability between the main network and the target network. The update of $\hat{\beta}$ is expressed as follows:

$$\hat{\beta}_c \leftarrow \omega\beta_c + (1-\omega)\hat{\beta}_c, \forall c = 1, 2, \tag{35}$$

where ω denotes the soft update ratio, and c is the number of network combinations.

5.2.2. Soft policy improvement

This approach involves the utilization of an energy-based model (EBM) alongside reparameterization techniques to update the actor. Here, the actor refines the network parameters by minimizing the Kullback–Leibler (KL) divergence between the Gaussian sampling policy π_α and the EBM. The goal is to bring π_α as close as possible to the EBM. Consistent with [44], The update function of α is expressed as $F_\pi(\alpha) = \mathbb{E}_{s_t\sim D, a_t\sim\pi_\alpha}\left[\varepsilon \log\pi_\alpha(a_t|s_t) - Q_\beta(s_t, a_t)\right]$.

Given the highly nonlinear Gaussian sampling process involved in the strategy, direct gradient propagation becomes infeasible. Accordingly, the form of the action must undergo transformation via reparameterization to separate sampling from gradient propagation. The update of α is denoted using reparameterization as follows:

$$F_\pi(\alpha) = \mathbb{E}_{s_t\sim D, \varepsilon\sim N}\left[\varepsilon \log\pi_\alpha\left(f_\alpha(\cdot)|s_t\right) - Q_\beta\left(s_t, f_\alpha(\cdot)\right)\right], \tag{36}$$

where $f_\alpha(\cdot) = f^\mu_\alpha(s_t) + n_t \odot f^\sigma_\alpha(s_t)$ represents a unit of Gaussian distribution sample. $f^\mu_\alpha(s_t)$ and $f^\sigma_\alpha(s_t)$ denote the mean and covariance of the strategy, respectively; and n_t is the noise sampled from a normal distribution. Consequently, the sampling process concerning the action is reformulated as a process with respect to n_t, which proves advantageous for gradient propagation and optimization.

5.2.3. Adaptive entropy regulation

The adaptively adjustable ε can accommodate complex environments and changing rewards, automatically adjusting the exploratory ability of the agent at different stages. The learning objective of MERL is reformulated as a constrained optimization problem, based on Eq. (31), to discover a stochastic policy with the highest expected reward while meeting a minimum expected entropy constraint, as follows:

$$\begin{aligned} (P_2): \quad & \max_\pi \mathbb{E}_{\rho_\pi}\left[\sum_{t=0}^{T} r(s_t, a_t)\right], \\ \text{s.t.} \quad & \mathbb{E}_{(s_t, a_t)\sim\rho_\pi}\left[-\log\left(\pi_t(a_t|s_t)\right)\right] \ge \mathrm{H}_0, \end{aligned} \tag{37}$$

where H_0 is the entropy threshold constant. When the offloading policy is exploring a new space, the optimal offloading action remains unclear, and the temperature should be increased to allow for further exploration. Once a certain space has been extensively explored, and the optimal offloading action is essentially determined, the temperature can then be reduced. The loss function of ε is denoted as follows by simplifying the equation using dynamic programming [44]:

$$F(\varepsilon) = \mathbb{E}_{a_t \sim \pi_t}\left[-\varepsilon \log \pi_t(a_t|s_t) - \varepsilon \mathrm{H}_0\right]. \tag{38}$$

Algorithm 1 ASTR

```
Input: Mini-batch sampling set $M$, parameters $\beta_1$, $\beta_2$, $\alpha$, and $\varepsilon$.
Output: Updated parameters $\beta_1$, $\beta_2$, $\alpha$, and $\varepsilon$.
 1: Initialize $\hat{\beta}_c \leftarrow \beta_c$, $c = \{1, 2\}$;
 2: Initialize the critic maximum update frequency $b_{\max}$ and the ASTR update factor $a \leftarrow 1$;
 3: for $b \leftarrow 1$ to $b_{\max}$ do
 4:     Randomly replace a transition in $M$;
 5:     Update $\beta_c$ by using Eq. (34) and $\beta_c = \beta_c - \lambda_\beta \hat{\nabla}_{\beta_c} F_Q(\beta_c)$;
 6:     Calculate $\tau$ according to Eq. (32);
 7:     if $a/b < 1/(2-\tau)$ then
 8:         Update $\varepsilon$ by using Eq. (38) and $\varepsilon = \varepsilon - \lambda_\varepsilon \hat{\nabla}_\varepsilon F(\varepsilon)$;
 9:         Update $\alpha$ by using Eq. (36) and $\alpha = \alpha - \lambda_\alpha \hat{\nabla}_\alpha F_\pi(\alpha)$;
10:         break;
11:     end if
12: end for
13: Update $\hat{\beta}_c$ by using Eq. (35).
```

Algorithm 2 STS-UDCO

```
Input: Environment observation $s_t$, number of episodes $e_{\max}$, mini-batch size $MB$, time sequence length $t_{\max}$.
Output: Optimum CO strategy.
 1: Initialize network parameters $\beta_1$, $\beta_2$, $\alpha$, $\varepsilon$, and replay memory $D = \emptyset$;
 2: Initialize two critic networks $Q_\beta$, two critic target networks $Q_{\hat{\beta}}$, and an actor network $\pi_\alpha$;
 3: for $e \leftarrow 1$ to $e_{\max}$ do
 4:     Reset environment parameters and get initial observation $s_0$;
 5:     Sort the DAG to task sets by using Eqs. (24) and (25);
 6:     for $t \leftarrow 1$ to $t_{\max}$ do
 7:         Obtain the observation $s_t$ from environmental parameters;
 8:         Perform $OBN$ on $s_t$ by using Eq. (28);
 9:         Obtain $a_t = \{\mathrm{UD}_i, o_u(t), v_u(t), \gamma_m(j,n)\} \sim \pi_\alpha(a_t|s_t)$;
10:         Update $L_u(t)$ by using Eqs. (2) and (3);
11:         if $L_u(t)! = L_u(t-1)$ then
12:             Calculate $E_f(t)$ by using Eq. (10);
13:         end if
14:         Calculate $E_h(t)$ by using Eq. (11);
15:         Calculate $\mathrm{TCC}_{j,n}$ by using $\gamma_m(j,n)$ and Eq. (23);
16:         Obtain $r(s_t, a_t)$ by using Eq. (30) and next state $s_{t+1}$;
17:         Store the tuple $(s_t, a_t, r_t, s_{t+1})$ into $D$;
18:     end for
19:     Randomly sample the Mini-batch $MB$ from $D$;
20:     Update $\beta_1$, $\beta_2$, $\alpha$, and $\varepsilon$ by using Algorithm 1;
21: end for
```

5.3. Description of STS-UDCO

This study combines ASTR with the SAC algorithm to devise a saturated training SAC-empowered UAV-assisted dependency-aware CO algorithm (STS-UDCO), to address P_1. The overarching framework of STS-UDCO comprises three primary components: an actor and a critic representing the intelligences and the UAV-assisted DAG-CO environment (Fig. 3). The actor and critic networks are constructed using multi-layer perceptions (MLPs) with two hidden layers. The actor's input corresponds to the observation state, while its output represents the mean and covariance of Gaussian-distributed policy actions, respectively. The critic's input comprises an empirical tuple, and its output yields the Q-value.

STS-UDCO comprises two primary phases: interaction and training phases. During the interaction phase, once the environment initialization is complete, the system transitions into a continuous operation state. At each time slot, the UAV inputs the current observation state collected into the actor network of the agent. Subsequently, the actor network generates a set of action spaces based on the current observation and policy. The UAV executes these action spaces to interact with the current environment, receiving reward feedback and the observation state for the next time slot. Thereafter, the agent stores the interaction tuple (s_t, a_t, r_t, s_{t+1}) into the buffer pool as an experience.

During the training phase, the agent samples a random batch of experiences from the buffer pool to provide to the actor and critic. The critic evaluates the selected action of the agent under ST, taking one experience (s_t, a_t, r_t, s_{t+1}) as input and outputting a value. Saturation training is achieved through ASTR and adaptive entropy temperatures, wherein the main Q network self-updates based on the sampling shifts and the target network and soft-updates the new parameters synchronized to the target network. Finally, $F_\pi(\alpha)$ completes the update based on the shift evaluation output by the main critic Q network.

The pseudo-code of the STS-UDCO algorithm is described in Algorithm 2. At the start of each time slot, STS-UDCO acquires state information from the design environment (lines 1–5). The interaction between the agent and the environment is depicted in lines 6–18. During this interaction, the UAV executes actions and adjusts its position based on the current policy. After the position update, the energy cost of the flight is computed. Thereafter, the TCC is calculated considering the optimal offload position for the task. The experience gained from this interaction is stored in the buffer pool. In lines 19–20, the buffer pool is sampled, and the network is updated based on the ASTR.

5.4. Complexity analysis

The space complexity of STS-UDCO is analyzed considering each interaction with the environment. The actor network generates a tuple of temporary experiences (i.e., s_t, a_t, r_t, and s_{t+1}) and stores it. Additionally, the critic network utilizes an experience tuple for self-updating at each time slot. The size of the storage space temporarily occupied by STS-UDCO during operation is determined by the experience tuple. Consequently, the space complexity is expressed as $O(2|S| + |A| + |R|)$, where $|S|$, $|A|$, and $|R|$ denote the total dimensions of s_t, a_t, and r_t, respectively [11].

The computational complexity of STS-UDCO is subsequently analyzed. The primary task of STS-UDCO is to interact with the environment and improve strategies based on experience. The computational complexity mainly depends on the training complexity of the neural networks involved, which includes the actor network and all the critic networks. These networks are MLPs. In a combination of MLPs with a fixed number of hidden layers and neurons, the computational complexity of the backpropagation update method scales proportionally with the product of the input and output sizes [45]. In STS-UDCO, the actor network takes states as inputs and outputs actions, while the critic network takes states and actions as inputs and outputs reward values obtained from the fitting process. Therefore, the complexity of the actor network is $O(|S| \times |A|)$ and the complexity of critic network is $O(|S| \times |A| \times |R|)$. Consequently, the computational complexity of STS-UDCO is expressed as $O(|S| \times |A| \times |R|)$.

Finally, the time complexity of STS-UDCO is analyzed. Let E be the maximum number of training iterations. In this case, the time complexity of STS-UDCO is expressed as $O(E \times T \times |S| \times |A| \times |R|)$ [4].

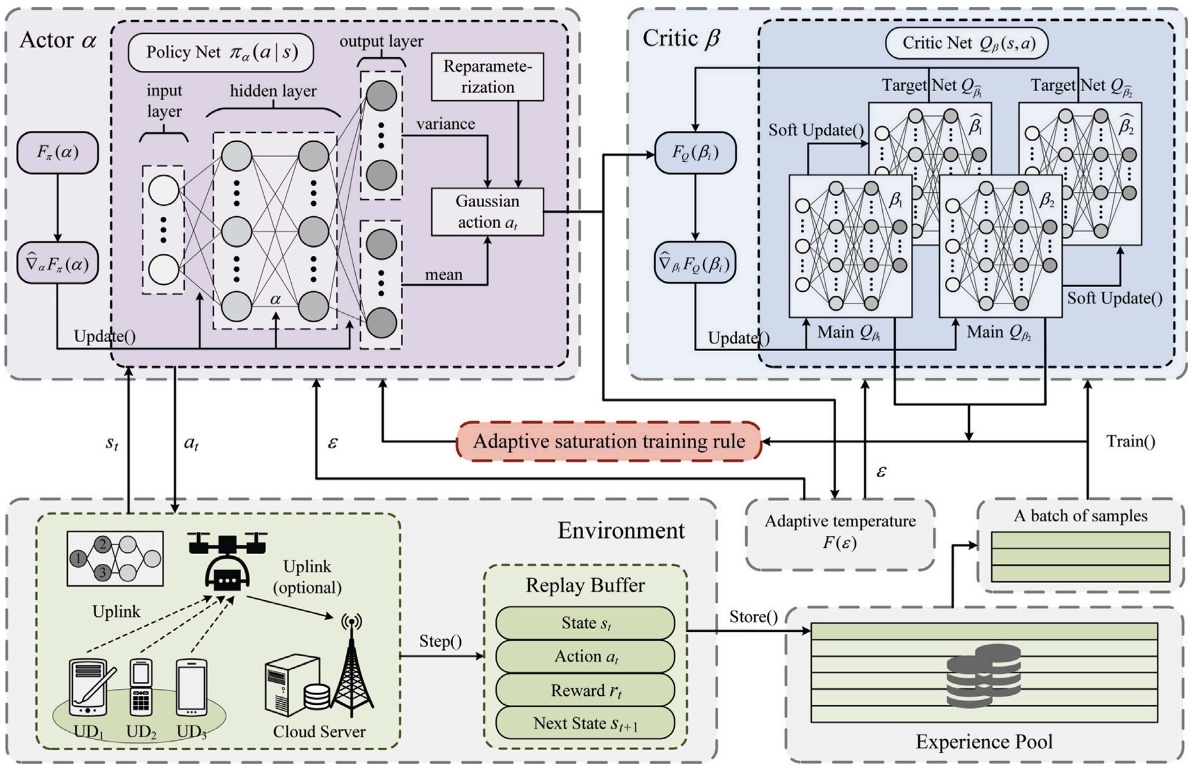

Fig. 3. Framework architecture of STS-UDCO.

6. Performance evaluation and analysis

In this section, a series of experiments is conducted to evaluate the performance of STS-UDCO. The experimental setup is first described, followed by an overview and discussion of the overall performance of STS-UDCO. A comparison is also made between STS-UDCO and other advanced algorithms using various DAG instances and CO environments.

6.1. Experimental setup

The experimental environment is built on the Python 3.9 platform, utilizing the Pytorch 1.12.0 machine learning library, and operates on Ubuntu 18.04. The UAV-assisted collaborative computation model considers a square area with a side length of 200 m. The initial position of the UAV is set at (100, 100, 100). The UDs are randomly generated at different ground positions within the service area. The cloud server is positioned outside the area at (300, 300, 0). The environment in this study focuses on time-varying channel states, where the presence of obstacles can affect the channel quality. The noise power of the impaired channel is set to a constant value for ease of processing. In these experiments, each neural network adopts an MLP architecture with one input layer, two hidden layers (with 256 and 64 nodes, respectively), and one output layer. The ReLU function is used as the activation function, and the Adam optimizer is utilized. The key experimental parameters are summarized in Table 3.

Four DAG instances are utilized in the experiments to simulate an application environment with task-dependent constraints [33,46]. These instances consist of 6, 12, 18, and 24 subtasks as indicated in Table 4. The structure of these instances is depicted in Fig. 1, where the subtasks are evenly distributed among the application entrance, application exit, and two intermediate layers. Sequential dependency order is established between the two neighboring layers, and each subtask can only be serviced once all the tasks in the preceding layer have been completed.

The following four algorithms are introduced for comparison to comprehensively evaluate the effectiveness of STS-UDCO:

Table 3
List of experimental parameters.

Parameter description	Value
Wireless bandwidth (B)	1 MH
Gaussian white noise (σ^2)	–100 dBm
Noise with obstacles (σ^2_{NLoS})	–80 dBm
Computing frequency of UD$_i$ (f_i)	1 GHz
Computing frequency of UAV (f_u)	5 GHz
Computing frequency of the cloud (f_c)	10 GHz
Channel gain per unit distance (h_0)	1×10^{-5}
Computational density (s)	1×10^{3} cycles/s
Uplink transmit power of UD$_i$ (P_i^{up})	1 W
Uplink transmit power of UAV (P_u^{up})	2 W
Power coefficient of CPU (κ)	1×10^{-28}
Battery capacity of UAV (E_b)	500 kJ

Table 4
Test instances.

Instance	Total number of subtasks
Instance 1	6
Instance 2	12
Instance 3	18
Instance 4	24

- SAC-based dynamic CO algorithm (SACDCO) [47]: This algorithm utilizes the original SAC algorithm to develop an optimal policy for UAV-assisted task offloading and resource allocation under resource constraints. Given that the objectives and MDP framework of Ref. [47] differ from those of STS-UDCO, we align the objectives of STS-UDCO with SACDCO while preserving the original MDP framework.
- DDPG-based CO algorithm (DDPGCO) [37]: This approach aims to minimize task processing delay while adhering to discrete variables and energy consumption constraints. DDPGCO utilizes a DDPG-based RL algorithm to address the CO problem. The experimental setup partially resembles that of DDPGCO.

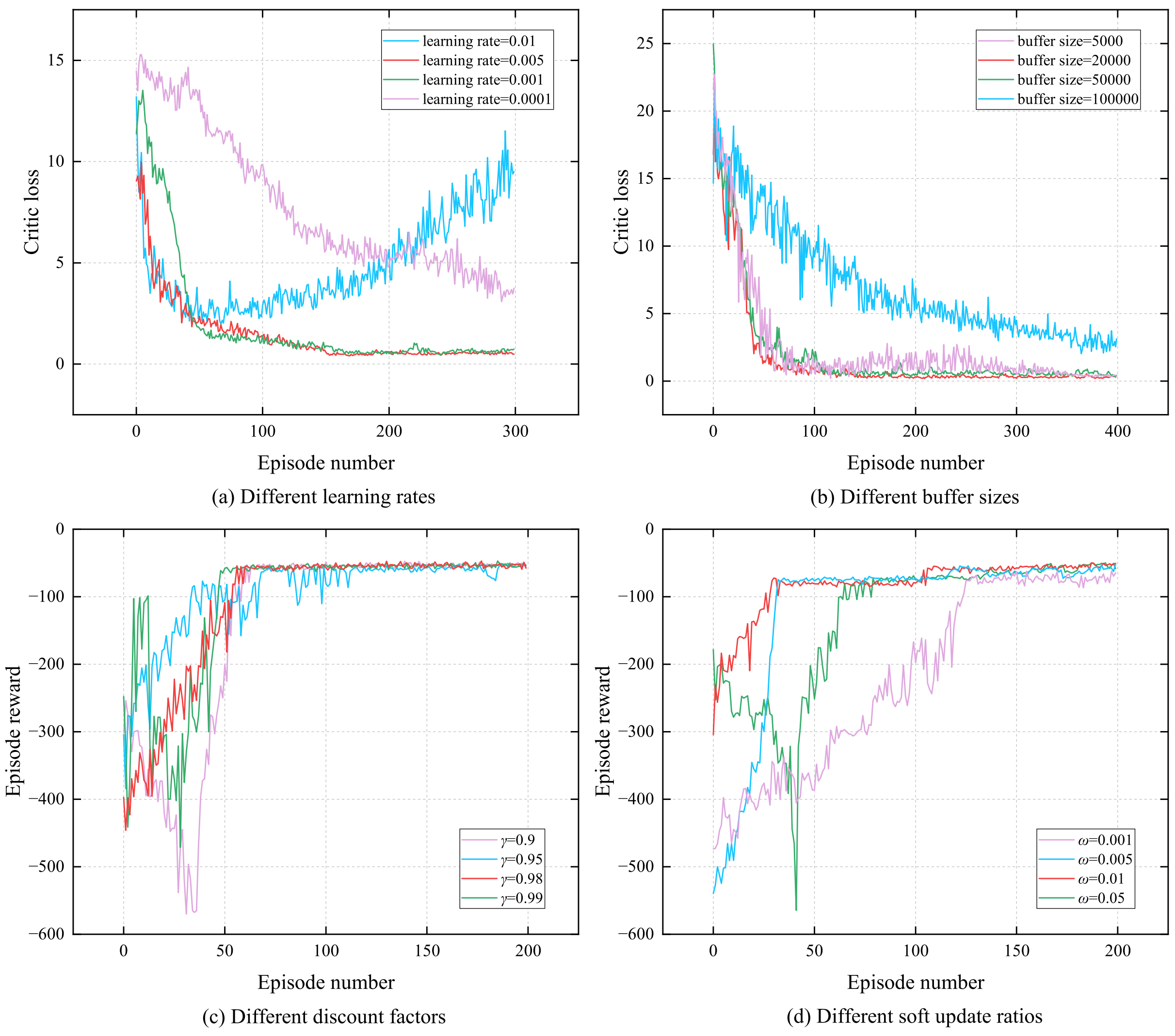

Fig. 4. Performance of loss function or episode reward of STS-UDCO with different hyperparameters.

- DQN-based DAG task scheduling algorithm (DQNDO) [48]: This scheme introduces a deep Q-network (DQN)-based optimization algorithm for the efficient task offloading in DAG and UAV deployment. The primary objective is to minimize the combined weighted system delay and energy consumption. Discretization operations are applied to align DQN with the MDP employed in this study.
- Local_only algorithm: In this algorithm, the minimization of costs associated with UAVs and cloud servers is ensured by retaining all tasks for local processing at the UDs.
- Edge_only algorithm: This algorithm offloads all tasks to UAV processing along predetermined flight routes, emphasizing the importance of the cloud server.

6.2. Analysis and setting of hyperparameters

This section investigates the influence of the various hyperparameters on the convergence and stability of STS-UDCO. Specifically, the analysis focuses on the LR of the critic network, the reward discount factor, the size of the experience pool, and the soft update scale factor. Experiments are conducted to identify the optimal values for these hyperparameters.

The influence of MSE LR on the loss function of the critic network is first examined. LR is varied within the range of 0.0001 to 0.01, leveraging the loss gradient to regulate the network weights and control the parameter updating speed. LR's effect on the loss curve does not consistently correlate with convergence performance as illustrated in Fig. 4(a), the loss function can gradually and accurately reach the optimal solution while maintaining stable convergence by setting LR to 0.005. By contrast, an LR value of 0.001 or 0.0001 results in slow convergence to suboptimal solutions or failure to converge, respectively. An LR of 0.01 causes the loss function to enter an oscillatory state post-convergence and eventually diverge. This situation occurs because a smaller LR reduces the optimizer learning efficiency, whereas a larger LR induces model instability and misfit. Consequently, LR is set to 0.005.

Second, the influence of buffer size on STS-UDCO is explored. The buffer size is varied within the range of 5000 to 100,000. The buffer pool stores past agent interactions and undergoes continuous updates using a "first-in-first-out" mechanism to provide suitable samples for training. The convergence of the loss function becomes increasingly challenging with larger buffer sizes, as depicted in Fig. 4(b). Specifically, when the buffer size is set to 50,000 or 100,000, the loss function fails to converge to the optimal solution or may not converge at all, respectively. Setting the buffer size to 5000 results in severe oscillations in convergence during later stages. A buffer size of 20,000 demonstrates optimal convergence. This situation is attributed to larger buffer pools resulting in lower filling rates and sampling efficiency, thereby diluting critical sampling data. Smaller buffer pools affect training stability due

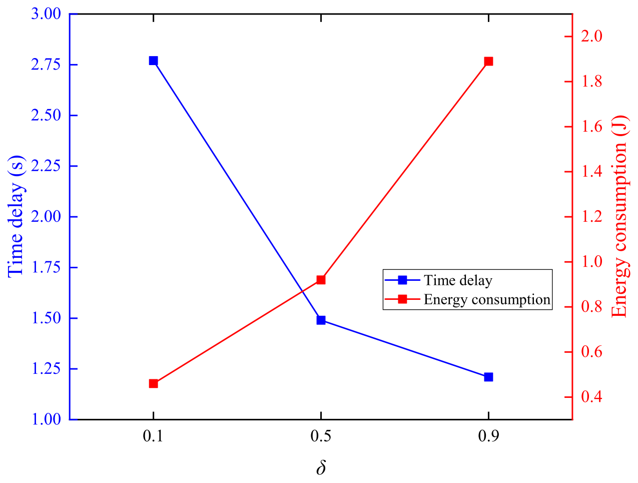

Fig. 5. Performance of time delay and energy consumption with different weights.

to high content turnover frequency. Consequently, the buffer size is set to 20,000.

Third, in the investigation of the discount factor γ in STS-UDCO, its range is set from 0.9 to 0.99. γ signifies the significance of future rewards to the agents in RL. Moreover, γ influences the convergence performance of iterative reward, as depicted in Fig. 4(c). Specifically, when γ is set to 0.95, the curve demonstrates oscillations in the late stage and struggles to converge to the optimal solution. When γ is set to 0.98, the reward robustly converges to the optimum. In the remaining two cases, the patterns diverge, with either large or small oscillations observed. This disparity arises because a smaller γ emphasizes unstable short-term benefits, whereas a larger γ disregards immediate rewards, resulting in inefficient training. Therefore, γ is set to 0.98.

At last, in the examination of the soft update factor ω in STS-UDCO, its range is established from 0.001 to 0.05. ω enhances the overall stability of the network by regulating the smoothness of soft updates in the target network. Moreover, ω influences the convergence performance of iterative reward, as depicted in Fig. 4(d). Specifically, when ω is set to 0.01, the curve demonstrates the most favorable convergence performance. When ω is set to 0.05, the curves oscillate widely, resulting in reverse optimization. The remaining two cases display varying degrees of sluggish convergence. This disparity arises because a smaller ω represents a lower percentage of the main network parameters, resulting in a slower update of the target network. In contrast, a larger ω affects the stability of the target network, which undermines the purpose of soft updating. Therefore, ω is set to 0.01.

6.3. Analysis of TCC weighting influence and task proportion

This section scrutinizes the weighting ratios of time delay and energy consumption in TCC. Moreover, this section explores the task allocation under the D–E–C three-tier architecture by employing TCC across each ratio. This experiment investigates the direction of task flow as the delay and energy consumption fluctuate with the weighting factors and analyzes the benefits of the three-tier computing architecture.

Fig. 5 illustrates the influence of the time delay weight factor δ on the relationship between time delay and energy consumption when set to 0.1, 0.5, and 0.9. To aid observation, the experiment uniformly scales down both time delay and energy consumption. The experiment uniformly scales down time delay and energy consumption to aid observation. Time delay and energy consumption are affected by varying magnitudes with the changing weight factors, with energy consumption demonstrating more fluctuations (Fig. 5). When δ is set to 0.1, the system prioritizes energy consumption optimization. The total energy consumption of a single task is reduced to 0.5. Time delay receives significant attention, as δ increases. A significant proportion of tasks prefer allocating higher energy to the server for computational purposes, resulting in a 3.8-fold increase in energy consumption and a 0.48-fold decrease in delay. Extreme weight settings can be utilized to fulfill offloading requests in exceptional cases, such as computation-intensive or delay-sensitive tasks. When both weighting factors are set to 0.5, the TCC achieves its minimum value, which is more advantageous for task offloading in uncommon scenarios.

In the UAV-assisted three-tier cooperative system, three devices with diverse computational capabilities are present: the UDs, the UAV on-board servers, and the cloud. Each device demonstrates distinct strengths in computation and incurs varying execution costs for CO. The unique execution advantages of these computing facilities result in alterations in the task distribution ratio across various experimental environments. However, how these alterations are manifested in the processing facilities for each task remains unclear. Statistical experiments are conducted to explore the proportion of tasks executed on each computing facility under different weight factors for delay and energy consumption to bridge this gap.

Fig. 6 illustrates the results of the statistical experiment. The proportion of tasks executed on the UDs decreases from an initial value of 30.7% to a negligible percentage with the gradual decrease in the weight factor δ. The UAV edge server initially handles the largest share of tasks when δ is set to 0.5, but its proportion decreases gradually. Correspondingly, the tasks assigned to cloud servers experience a gradual increase. These trends in performance are closely related to the computing power and execution cost of each computing device. When δ is set to 0.1, task offloading favors terminal devices and UAVs as the destination. This inclination stems from the higher transmission energy consumption associated with distant cloud servers. The energy consumption of the UAV is significantly reduced, by keeping tasks local, y keeping tasks local. The computing power and resources of UDs and UAVs may prove insufficient to meet stringent delay requirements with the gradual increase in δ. Despite the higher overall energy consumption of the cloud server, its robust computing capability enables it to provide high-speed computing services. When the weight factors for time delay and energy consumption are equal, the UAV, benefiting from its proximity to users, can flexibly provide convenient, fast, and energy-efficient computing services, making it the most suitable offloading destination at that point. This experiment demonstrates that the three computing devices with distinct capabilities can effectively handle CO requests under various requirements. The D–E–C fusion approach shows high reliability across diverse demand environments.

6.4. Instance performance and overall convergence analysis

This section applies STS-UDCO and five other comparison algorithms to four different DAG applications with identical environmental parameter configurations. The goal is to assess the overall offloading performance of these algorithms in diverse subtask-dependent environments. Additionally, the discrepancy in convergence performance between STS-UDCO and the comparison algorithms is evaluated throughout the entire training process.

Fig. 7 illustrates the system cost, which combines TCC and UAV energy consumption, across four instances utilizing various algorithms. All instances maintain consistent environmental configurations with the same total number of subtasks and data volume to mitigate the influence of varying user numbers on the experiment. Consequently, the percentage of tasks in the environment belonging to the same level of dependency order positions, excluding the start and end tasks, increases. This leads to a higher density of parent tasks that must be processed before child node tasks. Consequently, the UAV experiences reduced complexity and signaling in service delivery due to a higher proportion of tasks sharing identical dependency states. This expanded set of options grants the UAV greater flexibility, resulting in a gradual

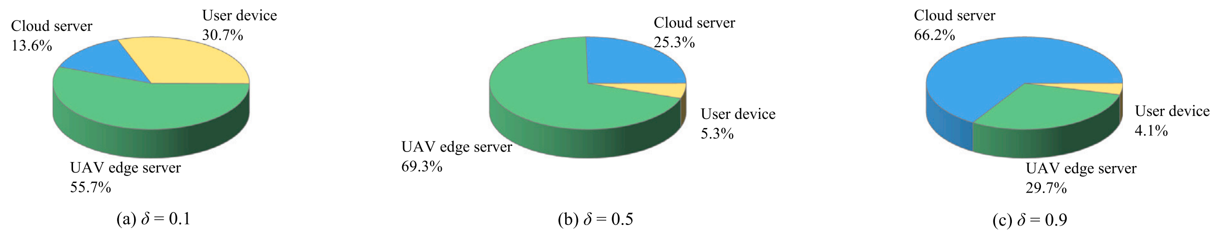

Fig. 6. Proportion of tasks executed on three computing devices.

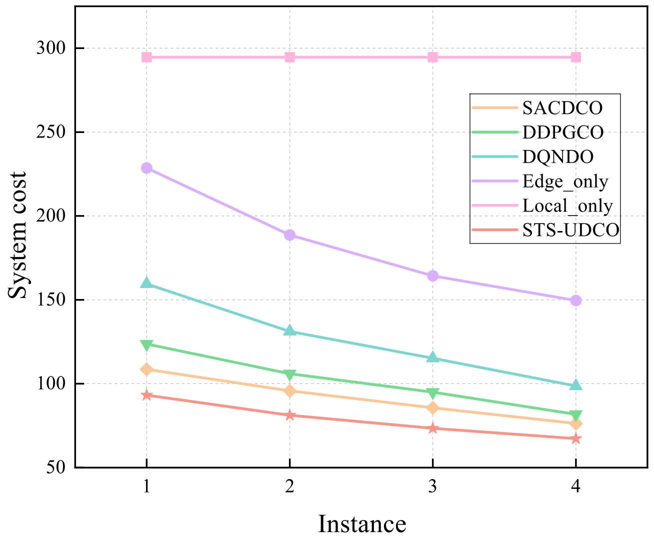

Fig. 7. System cost of different algorithms for different DAG instances.

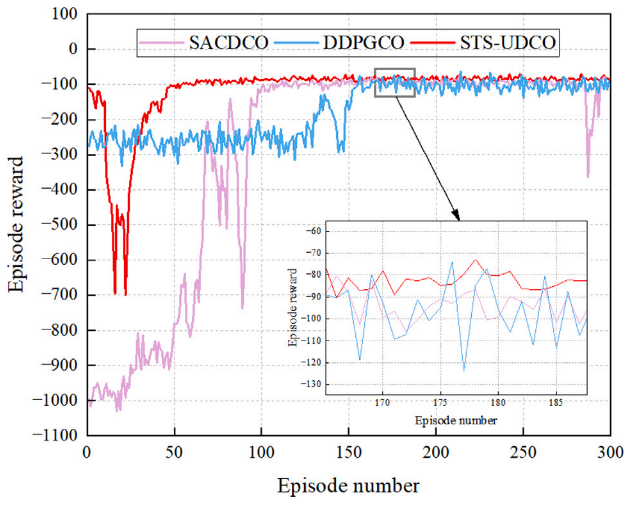

Fig. 8. Comparison of overall convergence performance with different algorithms.

decline in the overall task offloading cost. The Local_only algorithm persists in a consistently high-cost state as it exclusively executes tasks locally. By contrast, the DRL-based algorithm exhibits markedly lower system costs compared with Local_only and Edge_only algorithms. This discrepancy stems from the collaborative computing framework and the DRL-based offloading strategy, which proficiently optimizes the CO decision. The STS-UDCO algorithm achieves the most favorable system cost across diverse instances, demonstrating remarkable stability by minimizing performance differentials among them.

Fig. 8 showcases the reward training convergence curves for the three algorithms within the same example. The DDPGCO algorithm adheres to the "explore first, train later" principle, resulting in the initial fluctuations within a constrained range of rewards. This phenomenon arises from the early learning stages of the algorithm, during which the experience pool remains incomplete and the gathered samples possess limited value. However, the convergence efficiency steadily enhances as the experience pool accumulates additional valuable insights. The integration of Figs. 7 and 8 reveals that STS-UDCO has the swiftest convergence rate and the most precise convergence outcomes among the DRL-based algorithms. STS-UDCO consistently converges around 71 generations with the highest cumulative reward. STS-UDCO enhances convergence speed by 29.71% and 54.21%, and the optimal solution by 13.69% and 22.65% compared with SACDCO and DDPGCO, respectively. These enhancements stem from the incorporation of SAC in STS-UDCO, integrating policy entropy into rewards and optimizing policy generation via the AC architecture. This approach empowers the algorithm to explore various extremes of the model, maximizing its potential to identify the optimal solution. These advantages facilitate the early convergence of STS-UDCO. The decline in rewards observed during the pre-convergence period in Fig. 8 signifies the consideration of strategy entropy for exploration. STS-UDCO demonstrates superior convergence performance in contrast to SACDCO algorithms, which are part of the same SAC algorithm family. This contrast is evident in SACDCO's tendency to oscillate during convergence, ultimately failing to select the optimal solution. Two primary factors contribute to this discrepancy. First, the computational model's high-dimensional state and the multi-objective optimization MDP present challenges in network updating and convergence. However, ASTR effectively addresses this challenge by balancing updates between the actor and critic, dynamically adjusting their relative update frequencies to mitigate issues of critic misfitting. Second, STS-UDCO features a more refined MDP design, incorporating more specific observations, a well-designed action space, and the utilization of OBN and action mapping pairs. Experimental results highlight the greater stability of ASTR in converging toward the optimal solution.

6.5. *Comparative experiments in different CO environments*

In this section, four experiments are designed to verify the superiority of STS-UDCO and other advanced algorithms across various environments and task configurations.

6.5.1. *Influence of different task sizes*

The size of randomly generated task data directly affects the cost of the system and the selection of the CO destination. Larger task sizes exert greater pressure on the computing facility. In our experiments, tasks are classified as small (1.5–3 MB), medium (3–4.5 MB), and large (4.5–6 MB) to compare the TCC of each algorithm. As shown in Fig. 9(a) and (b), STS-UDCO demonstrates the lowest time delay and energy consumption among the five benchmark algorithms across different task sizes, followed by SACDCO, DDPGCO, and DQNDO. These algorithms are all based on DRL approaches, which gradually learn optimal offloading decisions through trial and error, resulting in improved performance compared with the Local_only and Edge_only algorithms. Moreover, the difference in energy consumption between

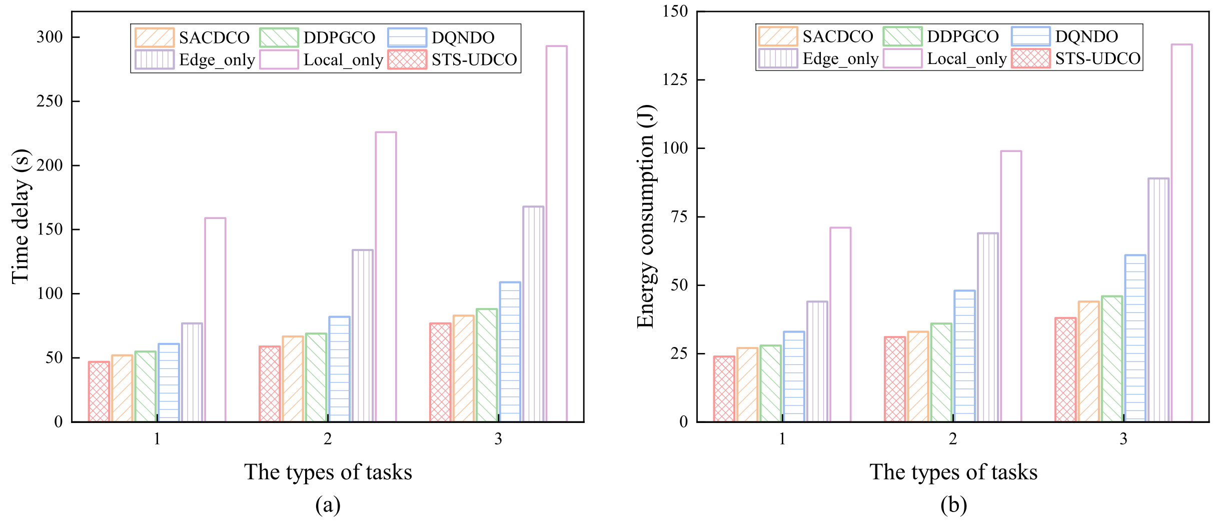

Fig. 9. Comparison on the average (a) time delay and (b) energy consumption with different task sizes.

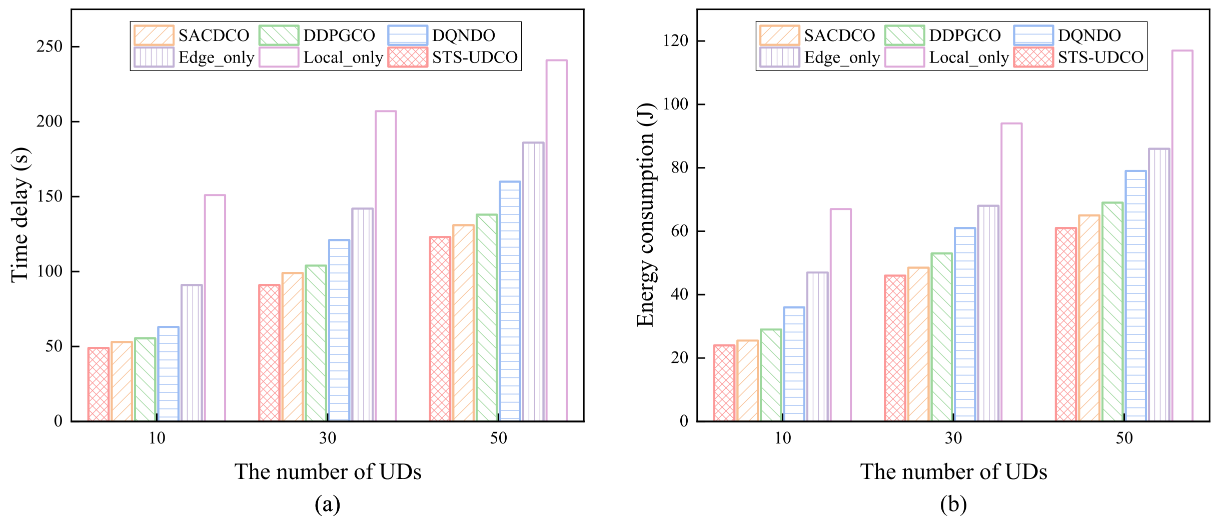

Fig. 10. Comparison on the average (a) time delay and (b) energy consumption with different numbers of UDs.

Local_only and the other algorithms is smaller compared with the difference in time delay as local processing is more energy-efficient. STS-UDCO, based on the ASTR-enhanced SAC framework, exhibits enhanced exploration and convergence capabilities, resulting in a 12.51% improvement in time delay and a 13.74% enhancement in energy consumption over the suboptimal algorithm for large-scale tasks.

6.5.2. Influence of different numbers of UDs

The increasing numbers of UDs requesting computing services has a significant influence on the optimization efficiency of the algorithms because of a higher volume of urgent tasks throughout the time series and more complex offloading decisions. In our experiments, we vary the number of UDs as 10, 30, and 50 to compare the TCC of each algorithm. The task density in the overall environment rises with the increase in the number of UDs, resulting in longer time delays and higher energy overhead (Fig. 10(a) and (b)). The experimental results demonstrate that STS-UDCO achieves the lowest system cost across different UD counts, outperforming the five advanced algorithms. This is attributed to the optimal solution capture capability of adaptive strategy entropy. In a massive user-distributed environment, STS-UDCO surpasses the next best algorithm by 9.56% and 12.86% in system cost, respectively.

6.5.3. Influence of the different bandwidths

The channel bandwidth plays a crucial role in task transmission efficiency and has a significant influence on UAVs that perform computational and transmission tasks. In our experiments, we vary the channel bandwidth as low (1 MHz), medium (2 MHz), and high (3 MHz) bandwidth to evaluate the time delay and energy consumption (i.e., TCC) of each algorithm. The TCC decreases with the increase in the channel bandwidth (Fig. 11(a) and (b)). Meanwhile, the TCC of the Local_only algorithm remains constant because it processes all local tasks and is unaffected by the bandwidth. The experimental results consistently demonstrate that STS-UDCO outperforms the five comparison algorithms across different bandwidth specifications. In a low bandwidth environment, STS-UDCO exhibits superior performance compared with the next best algorithm, SACDCO. Specifically, STS-UDCO achieves a 10.10% reduction in time delay and a 14.71% decrease in energy consumption.

6.5.4. Influence of different CPU processing powers

The efficiency of task computation, which refers to the number of processor cycles required to process 1 bit of data, is directly influenced by the CPU processing power. Multi-threading technology has the potential to significantly enhance CPU processing efficiency. In our

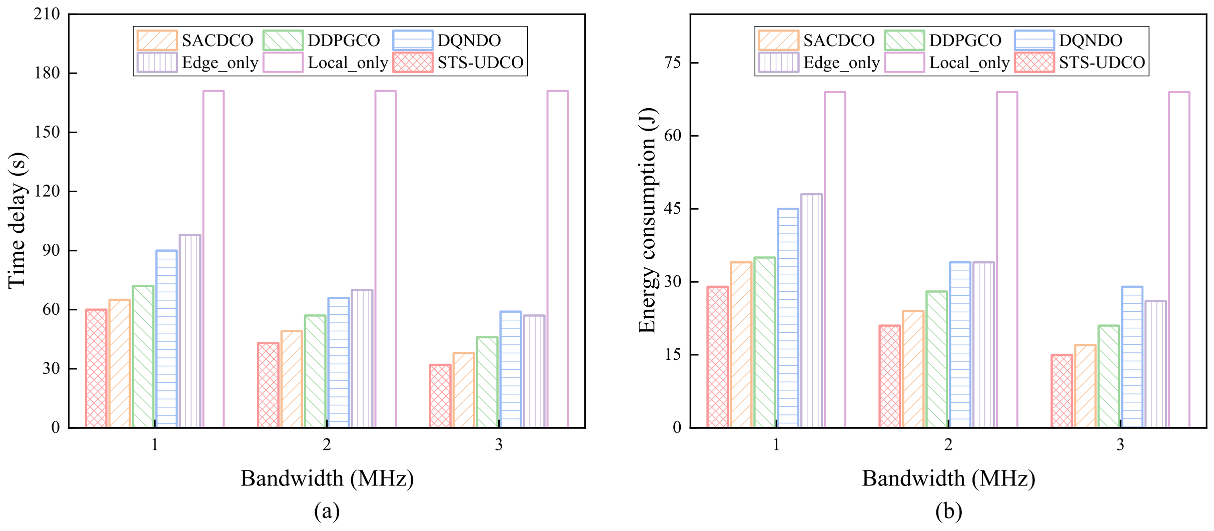

Fig. 11. Comparison on the average (a) time delay and (b) energy consumption with different bandwidths.

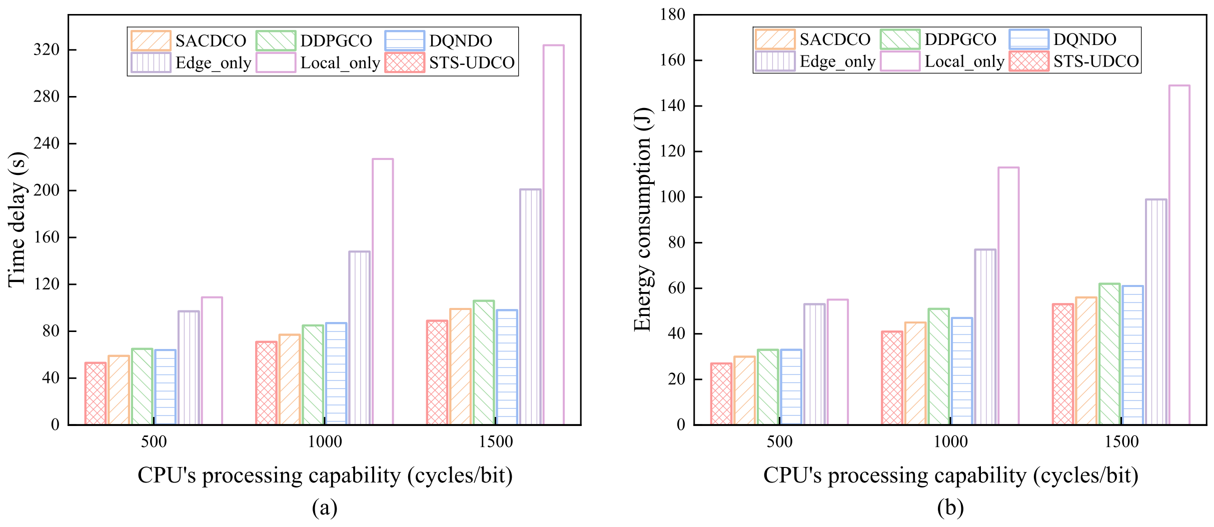

Fig. 12. Comparison on the average (a) time delay and (b) energy consumption with different CPU processing powers.

experiments, we vary the CPU processing power of the three types of computing facilities as strong (500 cycles/bit), normal (1000 cycles/bit), and weak (1500 cycles/bit) to evaluate the effect of different CPU processing powers. Fig. 12(a) and (b) illustrate the observation that time delay and energy consumption increase with the decrease in the processing power. The experimental results consistently demonstrate that STS-UDCO achieves optimal optimization effects under different CPU processing powers compared with the five comparison algorithms. STS-UDCO outperforms the suboptimal SACDCO algorithm by 13.60% and 8.63% at weak processing power, respectively. The DQNDO algorithm fails to properly converge at weak processing power. This situation arises because the reduced processing power causes a significant number of tasks to linger, exceeding the optimization limit of the DQNDO algorithm after the action discretization process. In our experiments, we enhance the weak processing capacity by increasing the other system resources of the DQNDO algorithm.

In summary, STS-UDCO demonstrates superior solving ability and stability performance in all four distinct environments. This feature is evident from its lowest levels of time delay and energy consumption and minimal performance variance across different loads, particularly in high load scenarios.

7. Conclusion and future work

This study focuses on addressing the dependency-aware CO problem in a UAV-assisted D–E–C cooperative computing system. To efficiently solve the problem of minimizing UAV battery consumption and TCC, we propose an algorithm called STS-UDCO and utilizes a specific MDP model to handle high-dimensional states and hybrid actions. We also introduce an ASTR in STS-UDCO to enhance the stability of the AC structure. ASTR dynamically encourages the critic to self-train based on the current fitting degree. Experimental results conducted on various instances and environments validate the effectiveness of STS-UDCO. In comparison with the other advanced algorithms, STS-UDCO significantly reduces system cost by at least 11.83% and optimizes delay and energy consumption in different environments by up to 13.60% and 14.71%, respectively. Meanwhile, STS-UDCO exhibits superior convergence and stability, with a maximum convergence speed of 39.10% and a convergence amplitude optimization effect of 67.39%, respectively.

However, this study primarily operates under the assumption of a single UAV environment and the absence of clear task deadlines. A basic UAV-assisted computing architecture presents notable security risks and vulnerability to interference. Furthermore, the utilization of UAV clusters and task queuing systems in actual offloading scenarios can significantly enhance the overall system performance.

In the near future, our research will focus on further investigating the CO and trajectory optimization problem based on multi-UAV co-operation and competition. We will also integrate pricing and security considerations into the CO strategies, enhance existing task models, and develop an offloading model that is better suitable for real-world utility scenarios.

CRediT authorship contribution statement

Longxin Zhang: Writing – review & editing, Writing – original draft, Validation, Methodology, Investigation, Funding acquisition, Conceptualization. **Runti Tan:** Writing – original draft, Software, Resources. **Yanfen Zhang:** Validation, Software, Resources. **Jiwu Peng:** Validation, Methodology, Data curation. **Jing Liu:** Validation, Methodology, Data curation. **Keqin Li:** Supervision, Methodology, Conceptualization.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The source code that support the findings of this study are available in/from hyperlink to data source https://github.com/TanRunti/JSA-STSUDCO.

Acknowledgments

The authors would like to thank five anonymous reviewers for their suggestions to improve the manuscript. The preliminary version of this paper has been submitted and accepted for presentation at the 2023 IEEE 29th International Conference on Parallel and Distributed Systems (ICPADS). This work was partially funded by the Natural Science Foundation of Hunan Province (Grant No. 2023JJ50204), the Scientific Research Foundation of Hunan Provincial Education Department, China (Grant No. 23B0560), the National Key R&D Program of China (Grant No. 2018YFB1003401), the National Natural Science Foundation of China (Grant Nos. 61702178, 62072172), and the Hunan Provincial Innovation Foundation For Postgraduate, China (Grant No. CX20241108).

References

[1] P. Lin, Q. Song, F.R. Yu, D. Wang, L. Guo, Task offloading for wireless VR-enabled medical treatment with blockchain security using collective reinforcement learning, IEEE Internet Things J. 8 (21) (2021) 15749–15761.

[2] L. Zhang, M. Ai, K. Liu, J. Chen, K. Li, Reliability enhancement strategies for workflow scheduling under energy consumption constraints in clouds, IEEE Trans. Sustain. Comput. 9 (2) (2024) 155–169.

[3] J. Shi, C. Li, Y. Guan, P. Cong, J. Li, Multi-UAV-assisted computation offloading in DT-based networks: A distributed deep reinforcement learning approach, Comput. Commun. 210 (2023) 217–228.

[4] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, D. Niyato, Multi-agent deep reinforcement learning for task offloading in UAV-assisted mobile edge computing, IEEE Trans. Wireless Commun. 21 (9) (2022) 6949–6960.

[5] C. Sun, W. Ni, X. Wang, Joint computation offloading and trajectory planning for UAV-assisted edge computing, IEEE Trans. Wireless Commun. 20 (8) (2021) 5343–5358.

[6] F. Xu, S. Wang, W. Su, L. Zhang, Joint optimization task offloading and trajectory control for unmanned-aerial-vehicle-assisted mobile edge computing, Comput. Electr. Eng. 111 (2023) 108916.

[7] D. Ebrahimi, S. Sharafeddine, P.-H. Ho, C. Assi, Autonomous UAV trajectory for localizing ground objects: A reinforcement learning approach, IEEE Trans. Mob. Comput. 20 (4) (2021) 1312–1324.

[8] A.V. Savkin, C. Huang, W. Ni, Joint multi-UAV path planning and LoS communication for mobile-edge computing in IoT networks with RISs, IEEE Internet Things J. 10 (3) (2023) 2720–2727.

[9] K. Li, Heuristic task scheduling on heterogeneous UAVs: A combinatorial optimization approach, J. Syst. Archit. 140 (2023) 102895.

[10] R. Jia, K. Zhao, X. Wei, G. Zhang, Y. Wang, G. Tu, Joint trajectory planning, service function deploying, and DAG task scheduling in UAV-empowered edge computing, Drones 7 (7) (2023) 443.

[11] A. Sacco, F. Esposito, G. Marchetto, P. Montuschi, Sustainable task offloading in UAV networks via multi-agent reinforcement learning, IEEE Trans. Veh. Technol. 70 (5) (2021) 5003–5015.

[12] H. Yang, J. Zhao, Z. Xiong, K.-Y. Lam, S. Sun, L. Xiao, Privacy-preserving federated learning for UAV-enabled networks: Learning-based joint scheduling and resource management, IEEE J. Sel. Areas Commun. 39 (10) (2021) 3144–3159.

[13] H. Zhou, K. Jiang, X. Liu, X. Li, V.C.M. Leung, Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing, IEEE Internet Things J. 9 (2) (2022) 1517–1530.

[14] L. Zhang, R. Tan, M. Ai, H. Xiang, C. Peng, Z. Zeng, DSUTO: Differential rate SAC-based UAV-assisted task offloading algorithm in collaborative edge computing, in: 2023 IEEE 29th International Conference on Parallel and Distributed Systems, ICPADS, 2023, pp. 2329–2336.

[15] Y. Sahni, J. Cao, L. Yang, Y. Ji, Multi-hop multi-task partial computation offloading in collaborative edge computing, IEEE Trans. Parallel Distrib. Syst. 32 (5) (2021) 1133–1145.

[16] Y. Sun, Z. Wu, K. Meng, Y. Zheng, Vehicular task offloading and job scheduling method based on cloud-edge computing, IEEE Trans. Intell. Transp. Syst. 24 (12) (2023) 14651–14662.

[17] G. Wu, H. Wang, H. Zhang, Y. Zhao, S. Yu, S. Shen, Computation offloading method using stochastic games for software-defined-network-based multiagent mobile edge computing, IEEE Internet Things J. 10 (20) (2023) 17620–17634.

[18] Y. Ding, K. Li, C. Liu, K. Li, A potential game theoretic approach to computation offloading strategy optimization in end-edge-cloud computing, IEEE Trans. Parallel Distrib. Syst. 33 (6) (2022) 1503–1519.

[19] G. Wu, X. Chen, Y. Shen, Z. Xu, H. Zhang, S. Shen, S. Yu, Combining Lyapunov optimization with actor–critic networks for privacy-aware IIoT computation offloading, IEEE Internet Things J. 11 (10) (2024) 17437–17452.

[20] H. Zhou, Z. Wang, H. Zheng, S. He, M. Dong, Cost minimization-oriented computation offloading and service caching in mobile cloud-edge computing: An A3C-based approach, IEEE Trans. Netw. Sci. Eng. 10 (3) (2023) 1326–1338.

[21] L. Zeng, Q. Liu, S. Shen, X. Liu, Improved double deep q network-based task scheduling algorithm in edge computing for makespan optimization, Tsinghua Sci. Technol. 29 (3) (2024) 806–817.

[22] J. Guo, G. Huang, Q. Li, N.N. Xiong, S. Zhang, T. Wang, STMTO: A smart and trust multi-UAV task offloading system, Inform. Sci. 573 (2021) 519–540.

[23] X. He, R. Jin, H. Dai, Multi-hop task offloading with on-the-fly computation for multi-UAV remote edge computing, IEEE Trans. Commun. 70 (2) (2022) 1332–1344.

[24] Y. Chen, B. Ai, Y. Niu, H. Zhang, Z. Han, Energy-constrained computation offloading in space-air-ground integrated networks using distributionally robust optimization, IEEE Trans. Veh. Technol. 70 (11) (2021) 12113–12125.

[25] M.H. Mousa, M.K. Hussein, Efficient UAV-based mobile edge computing using differential evolution and ant colony optimization, PeerJ Comput. Sci. 8 (2022) 870.

[26] S. Gong, M. Wang, B. Gu, W. Zhang, D.T. Hoang, D. Niyato, Bayesian optimization enhanced deep reinforcement learning for trajectory planning and network formation in multi-UAV networks, IEEE Trans. Veh. Technol. 72 (8) (2023) 10933–10948.

[27] B. Li, W. Liu, W. Xie, X. Li, Energy-efficient task offloading and trajectory planning in UAV-enabled mobile edge computing networks, Comput. Netw. 234 (2023) 109940.

[28] X. Wei, L. Cai, N. Wei, P. Zou, J. Zhang, S. Subramaniam, Joint UAV trajectory planning, DAG task scheduling, and service function deployment based on DRL in UAV-empowered edge computing, IEEE Internet Things J. 10 (14) (2023) 12826–12838.

[29] Z. Jiang, R. Cao, S. Zhang, Joint optimization strategy of offloading in multi-UAVs-assisted edge computing networks, J. Ambient Intell. Humaniz. Comput. 14 (2023) 4385–4399.

[30] Z. Peng, G. Wang, W. Nong, Y. Qiu, S. Huang, Task offloading in multiple-services mobile edge computing: A deep reinforcement learning algorithm, Comput. Commun. 202 (2023) 1–12.

[31] S. Shen, L. Xie, Y. Zhang, G. Wu, H. Zhang, S. Yu, Joint differential game and double deep Q-networks for suppressing malware spread in industrial internet of things, IEEE Trans. Inf. Forensics Secur. 18 (2023) 5302–5315.

[32] L. Zhang, M. Ai, R. Tan, J. Man, X. Deng, K. Li, Efficient prediction of makespan matrix workflow scheduling algorithm for heterogeneous cloud environments, J. Grid Comput. 21 (4) (2023) 75.

[33] X. Liu, Z.-Y. Chai, Y.-L. Li, Y.-Y. Cheng, Y. Zeng, Multi-objective deep reinforcement learning for computation offloading in UAV-assisted multi-access edge computing, Inform. Sci. 642 (2023) 119154.

[34] S. Jeong, O. Simeone, J. Kang, Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning, IEEE Trans. Veh. Technol. 67 (3) (2018) 2049–2063.

[35] K. Dorling, J. Heinrichs, G.G. Messier, S. Magierowski, Vehicle routing problems for drone delivery, IEEE Trans. Syst. Man Cybern.: Syst. 47 (1) (2017) 7085.

[36] X. Xu, K. Liu, P. Dai, F. Jin, H. Ren, C. Zhan, S. Guo, Joint task offloading and resource optimization in NOMA-based vehicular edge computing: A game-theoretic DRL approach, J. Syst. Archit. 134 (2023) 102780.

[37] Y. Wang, W. Fang, Y. Ding, N. Xiong, Computation offloading optimization for UAV-assisted mobile edge computing: a deep deterministic policy gradient approach, Wirel. Netw. 27 (2021) 2991–3006.

[38] I. Budhiraja, N. Kumar, H. Sharma, M. Elhoseny, Y. Lakys, J.J.P.C. Rodrigues, Latency-energy tradeoff in connected autonomous vehicles: A deep reinforcement learning scheme, IEEE Trans. Intell. Transp. Syst. 24 (11) (2023) 13296–13308.

[39] G. Wu, Z. Xu, H. Zhang, S. Shen, S. Yu, Multi-agent DRL for joint completion delay and energy consumption with queuing theory in MEC-based iIoT, J. Parallel Distrib. Comput. 176 (2023) 80–94.

[40] L. Zhang, D. Liu, M. Ai, R. Tan, Z. Zeng, Reliability enhancement algorithm based on budget level in cloud-edge environments, Int. J. Embed. Syst. 16 (1) (2023) 9–22.

[41] V. Mnih, K. Kavukcuoglu, D. Silver, A.A. Rusu, J. Veness, M.G. Bellemare, Human-level control through deep reinforcement learning, Nature 518 (2015) 529–533.

[42] N. Shrivastava, M.A. Hanif, S. Mittal, S.R. Sarangi, M. Shafique, A survey of hardware architectures for generative adversarial networks, J. Syst. Archit. 118 (2021) 102227.

[43] X. Li, Y. Qin, J. Huo, W. Huangfu, Heuristically assisted multiagent RL-based framework for computation offloading and resource allocation of mobile-edge computing, IEEE Internet Things J. 10 (17) (2023) 15477–15487.

[44] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, S. Levine, Soft actor-critic algorithms and applications, 2018, arXiv e-prints, arXiv:1812.05905.

[45] M. Sipper, A serial complexity measure of neural networks, in: IEEE International Conference on Neural Networks, Vol. 2, 1993, pp. 962–966.

[46] F. Song, H. Xing, X. Wang, S. Luo, P. Dai, K. Li, Offloading dependent tasks in multi-access edge computing: A multi-objective reinforcement learning approach, Future Gener. Comput. Syst. 128 (2022) 333–348.

[47] S. Li, X. Hu, Y. Du, Deep reinforcement learning for computation offloading and resource allocation in unmanned-aerial-vehicle assisted edge computing, Sensors 21 (19) (2021) 6499.

[48] J. Li, Y. Pan, Y. Xia, Z. Fan, X. Wang, J. Lv, Optimizing dag scheduling and deployment for Iot data analysis services in the multi-UAV mobile edge computing system, Wirel. Netw. (2023).