Scilight

Journal of Machine Learning and Information Security
https://www.sciltp.com/journals/jmlis

Article

Joint Optimization for Dependency-Aware Computation-Offloading and Cooperative Service-Caching in Edge Computing

Yaoqian Gui and Fei Wang *

College of Electronic and Information Engineering, Southwest University, Chongqing 400715, China
* Correspondence: wsf0107@163.com

How To Cite: Gui, Y.; Wang, F. Joint Optimization for Dependency-Aware Computation-Offloading and Cooperative Service-Caching in Edge Computing. *Journal of Machine Learning and Information Security* **2026**, *2*(2), 11. https://doi.org/10.53941/jmlis.2026.100011

Received: 23 December 2025
Revised: 21 March 2026
Accepted: 27 March 2026
Published: 2 June 2026

Abstract: Dependency-aware computation-offloading and cooperative service-caching are pivotal technologies in mobile edge computing (MEC) systems. However, existing works often overlook the interdependencies among tasks/subtasks and the limited caching capacity of MEC servers, leading to suboptimal energy-time cost (ETC). In this paper, we investigate the joint optimization of computation-offloading, service-caching, and resource-allocation in an environment with multiple MEC servers, where multiple Internet of things devices (IoTDs) generate tasks composed of interdependent subtasks which are modeled as directed acyclic graphs (DAGs). Considering the constraints of limited caching capacity and computation resources at MEC servers, we formulate a mixed-integer nonlinear programming (MINLP) problem aimed at minimizing the weighted sum of task completion delay and energy consumption across all IoTDs, i.e., ETC. To solve this challenging problem, we design a scheme by integrating graph convolutional networks (GCN) with deep reinforcement learning (DRL), in which GCN extracts subtask features and dependencies among subtasks, and combines them with the features of the system environment as the state of our DRL method. We then employ an advantage actor-critic (A2C) framework with hybrid action space handling capability to perform Markov decisions and obtain optimal caching, offloading, and resource-allocation actions. Third, we conduct extensive experiments to evaluate the effectiveness of our proposed scheme. The simulations demonstrate that, compared with state-of-the-art baselines, our scheme achieves the lowest ETC in most cases, converges faster, and maintains robustness under diverse experimental conditions, including varying transmission rates, numbers of IoTDs, subtask counts, and caching capacity. Moreover, by conducting comparative analyses with relevant literature, we further confirm that our joint optimization framework has significant advantages in balancing energy efficiency and latency in the MEC environment.

Keywords: mobile edge computing (MEC); directed acyclic graph (DAG); computation-offloading; cooperative service-caching; graph convolutional networks (GCN); deep reinforcement learning (DRL)

1. Introduction

Recently, mobile edge computing (MEC) has been recognized as a transformative technique capable of significantly reducing the task offloading delay and energy consumption of Internet of Things devices (IoTDs) [1–5]. However, since MEC servers generally have limited computation and caching resources and there exists strong interference among IoTDs, IoTDs' energy-time cost (ETC) for task processing may be still very high. To resolve this issue, without considering task or subtask dependency, the existing works have widely studied the joint problem of computation-offloading and resource-allocation with/without caching resources optimization. Consider a typical smart surveillance scenario, where multiple IoTD cameras are deployed in a smart city. Each camera runs a real-time

video analytics application that involves multiple interdependent subtasks: video frame capture, motion detection, object recognition, and alert generation. However, in reality, there may exist certain dependencies, e.g., sequential dependency, among tasks or subtasks [6], which may deeply affect the schemes for computation-offloading, caching-allocation, and resource-allocation, and then affect IoTDs' ETC for task processing. Therefore, the joint optimization and design for dependency-aware computation-offloading, service-caching, and resource-allocation have become urgent issues but have not been well studied.

As mentioned above, to minimize IoTDs' ETC for task processing, many works, e.g., [7–13], have studied the problem of computation-offloading and/or resource-allocation. However, in [14, 15], the authors assumed that tasks cannot be partitioned or tasks can be partitioned but the dependencies among subtasks have not been taken into consideration. Therefore, the works [7–13] studied the problem of computation-offloading and/or resource-allocation for the scenarios when tasks can be partitioned and subtasks are mutually dependent, where each IoTD's subtasks are modeled as a directed acyclic graph (DAG). Specifically, in a DAG, the nodes stand for subtasks and the edges connected by nodes stand for the dependencies among subtasks, where subtasks are offloaded or executed according to the structural order of the DAG. In [8, 9], with considering subtask dependencies, the authors investigated the dependency-aware computation-offloading problem for the Internet of Vehicles. In [10], the authors focused on reducing application completion time via the efficient offloading of dependent tasks to resource-constrained MEC servers. Similarly, in [11, 12], the authors proposed the joint computation-offloading and resource-allocation schemes to minimize task delay and improve MEC servers' task processing efficiency. In [13], the authors designed a request service provision system according to the intelligent reinforcement learning to meet the resource and delay requirements of IoT applications. The authors of [16–18] all investigated the problem of dependency-aware computation-offloading for MEC scenarios with multiple users, aiming to minimize users' ETC for task processing.

The authors of [19], considering the dependency among tasks, proposed a graph-partition based storage scheme for DAG-blockchain, dynamically retaining high-freshness transactions at MEC servers to minimize data storage costs and enhance scalability in industrial IoT environments. Meanwhile, the authors of [20–22] all proposed DRL-based dependency-aware resource-allocation and computation-offloading schemes for the purpose of minimizing the weighted sum of delay, energy, and monetary cost, etc. However, the works [7–22] all assumed that necessary service programs for task processing have been pre-cached in MEC servers, which may not be consistent with the reality because MEC servers generally have limited caching capacity [23, 24].

In practice, an MEC server cannot cache all service programs demanded by IoTDs. The authors of [25] studied how to effectively improve the caching performance of multi-edge collaborative caching systems. The work [26] studied the issue of how to efficiently combine service-caching and resource scheduling in ultra-dense MEC networks. Accordingly, the authors of [27, 28] considered the joint optimization problem for computation-offloading and service-caching for MEC-enabled smart grids and vehicular edge computing systems, respectively. The authors of [29, 30] both studied content-caching, where the former studied collaborative content-caching and computation-offloading and the latter investigated the joint problem for content-caching, service placement, and computation-offloading for uncrewed aerial vehicles (UAV) communication systems. Although all of the aforementioned studies mentioned service-caching, they did not take into account the dependencies among tasks/subtasks.

Since task dependencies directly impact computation-offloading, caching-allocation, etc., the authors of [31] considered DRL-based UAV trajectory planning and dependent subtask scheduling for UAV-aided MEC systems, where a UAV collects tasks with dependent subtasks from multiple ground users. The authors of [32] considered dynamic content-caching and dependency-aware computation-offloading in MEC. However, when considering task/subtask dependencies, the joint optimization problem for computation-offloading, cooperative service-caching, and resource-allocation for MEC systems with multiple MEC servers serving multiple loTDs has been hardly studied.

To tackle the aforementioned challenges, we propose a joint optimization scheme for dependency-aware computation-offloading and cooperative service-caching for MEC systems with multiple MEC servers which cooperatively serve multiple IoTDs with the aid of a cloud. First, considering the constraints of limited caching capacity and computation resources of MEC servers, we formulate a mixed integer nonlinear programming problem to minimize ETC, by modeling the dependency relationship among subtasks through a DAG. Second, since traditional neural networks have difficulty in handling the offloading problem of subtasks with dependencies, we integrate graph convolutional networks (GCN) with deep reinforcement learning (DRL) to solve the formulated optimization problem. Specifically, we use GCN to capture and update the dependencies and features of subtasks (i.e., the data size of subtasks and the CPU period of subtask execution), and then combine them with the features of the MEC environment to obtain the state of DRL. Then, DRL is used to obtain the subtask offloading, service-caching, and resource-allocation decisions, construct the Markov process, and obtain the reward and the next state. Third, we

assess the performance of our proposed scheme via extensive simulations, showing the superiority of our proposed scheme in balancing energy efficiency and delay in the MEC environment.

The remainder of this paper is structured as follows. Section 2 establishes the system models. Section 3 develops the joint cooperative service-caching, computation-offloading, and resource-allocations schemes. Section 4 performs systematic simulation evaluations to evaluate our proposed schemes. Section 5 discusses the comparison between the literature and our paper. Section 6 serves as the concluding segment. We also compare our work in this paper with some related works in terms of algorithm, offloading, caching, dependency, limited C2 resource (computation resources and caching resources), and optimization objective in Table 1. Table 2 provides a specification of core system parameters.

Table 1. Comparison of solutions for related work.

Ref.	Algorithm	Offloading	Caching	Dependency	Limited C2 Resource	Optimization Objective
[14]	SAC	Binary	NA	×	✓	Minimizing the Age of Information subject to energy and costs.
[9]	ODCO	Binary	NA	✓	✓	Minimizing the ETC of all vehicle applications.
[16]	MO	Binary	NA	✓	✓	Minimizing the ETC of all applications.
[17]	COFE	Partial	NA	✓	×	Minimizing the average makespan.
[18]	GCN+PPO	Binary	NA	✓	✓	Minimizing system overhead.
[19]	GpDB	Binary	NA	✓	✓	Minimizing the storage cost of MEC servers.
[20]	TF-DDRL	Binary	NA	✓	✓	Minimizing the ETC and monetary costs.
[21]	X-DDRL	Binary	NA	✓	✓	Minimizing the ETC of all IoTDs.
[22]	μ-DDRL	Binary	NA	✓	✓	Minimizing the execution time of each service.
[31]	DQN	Binary	Service-caching	✓	✓	Maximizing DAG task processing efficiency in UAV operations.
[23]	DDPG	Partial	Service-caching	×	✓	Minimizing system energy.
[32]	DDPG	Partial	Content-caching	✓	✓	Minimizing the average delay.
Our	GCN+A2C	Binary	Service-caching	✓	✓	Minimizing the ETC for all applications.

ODCO: optimized distributed computation-offloading; MO: Mathematical Operation; COFE: online offloading framework for multiple mobile applications; GpDB: Graph-partition based storage strategy for DAG-Blockchain; TF-DDRL: Transformer-enhanced Distributed DRL scheduling technique.

Table 2. System variables.

Symbol	Description
$\mathcal{K}$	Set of all K IoTDs
$\mathcal{N}$	Set of all N MEC servers
$\mathcal{I}$	Set of I subtasks
G_k	Directed acyclic graph of IoTD k
V_k	Set of vertices of G_k
$\mathcal{E}_k$	Set of edges of G_k
$c_{k,n}$	Binary variable indicating whether the service required by IoTD k has been cached by MEC server n
$z^{\mathrm{L}}_{k,i}$	Binary variable indicating whether subtask i is executed locally at IoTD k
$z_{k,i,n}$, $z^{\mathrm{c}}_{k,i}$	Binary variable indicating whether subtask i of IoTD k is offloaded to MEC server n or the cloud server
$L_{k,i}$	Total number of CPU cycles required for processing subtask i of IoTD k
$D_{k,i}$	Size of input data related to subtask i of IoTD k
$f_{n,k}$	Computation resource of MEC server n allocated to IoTD k
f^{L}_k	Computation resource of IoTD k
$f_{\mathrm{c},k}$	Computation resource of the cloud center allocated to IoTD k
$T^{\mathrm{L}}_{k,i}$	Computation execution time of subtask i at IoTD k by local computing
T_{k,i,n_k}	Transmission time at IoTD k to offload data to its nearest MEC server n_k
$T_{k,i,m}$	Transmission time from MEC server n_k to MEC server m
$T^{\mathrm{c}}_{k,i}$	Transmission time from MEC server n_k to the cloud
$T^{\mathrm{off}}_{k,i,j}$	Transmission time for transmitting $u_{k,i,j}$ bits of results from IoTD k to MEC server n_k
R_{k,n_k}	Uplink transmission rate from device k to MEC server n_k
$R_{n_k,m}$	Transmission rate from MEC server n_k to MEC server m
$R_{n_k,\mathrm{c}}$	Transmission rate from MEC server n_k to the cloud
$E^{\mathrm{L}}_{k,i}$	Energy consumption for subtask i at IoTD k by local computing
E_{k,i,n_k}	Energy consumption at IoTD k for transmitting subtask i to MEC server n_k
$E^{\mathrm{off}}_{k,i,j}$	Transmission energy consumption for transmitting $u_{k,i,j}$ bits of results from IoTD k to MEC server n_k
$W_{n_k,k}$	Channel bandwidth allocated to IoTD k by MEC server n_k

2. The System Models

2.1. System Model

Consider an MEC system shown in Figure 1, including N *MEC servers*, indexed by $\mathcal{N} \triangleq \{1, 2, \ldots, n, \ldots, N\}$, K IoTDs, indexed by $\mathcal{K} \triangleq \{1, 2, \ldots, k, \ldots, K\}$, and a *cloud center*. The IoTDs are connected to MEC servers via wireless links, and MEC servers connect to each other and the cloud server all through the fiber wired links. Each IoTD needs to offload all or part of its task to MEC servers and/or the cloud server because of its limited computation and storage resources. Each MEC server is deployed at a base station (BS) and all MEC servers have different computation and storage resources. Different MEC servers utilize the same spectrum, while IoTDs connected to the same MEC server utilize orthogonal uplink spectrums. The cloud server serves as the centralized fallback processing unit with two key assumptions: (i) it possesses sufficient computational resources (i.e., $f_{c,k}$ is considered sufficiently large to handle any offloaded subtask); and (ii) it is assumed to have cached all required service programs for all IoTDs. Therefore, if a subtask cannot be processed locally at the IoTD or at any cooperative MEC server (due to lack of cached services or insufficient computation resources), it is ultimately offloaded to the cloud.

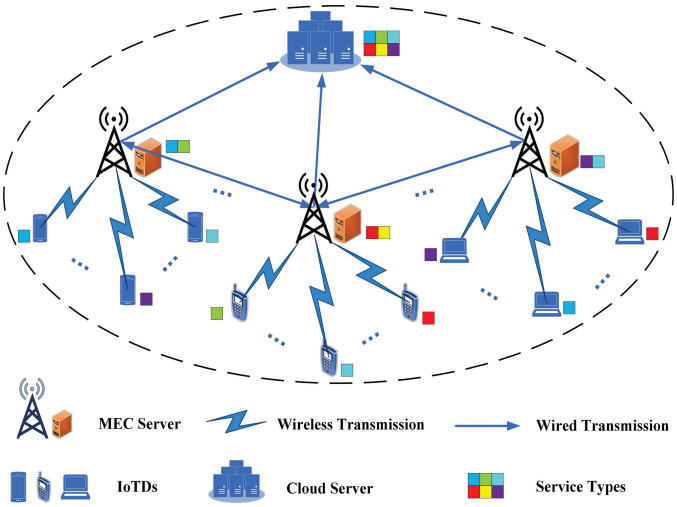

Figure 1. Framework of our proposed MEC system.

Each IoTD k has a computation task, which can be divided into I dependency subtasks, indexed by the set $\mathcal{I} \triangleq \{1, 2, \ldots, i, \ldots, I\}$. The computational task of each IoTD is segmented into multiple subtasks according to the functional modules of the application and their data dependencies. Each subtask performs a specific operation and requires input data from its predecessor subtasks, forming a DAG structure. This segmentation enables flexible and dependency-aware offloading decisions at the subtask level. These subtasks can be executed at IoTDs or be offloaded to MEC servers or the cloud center. Moreover, since each MEC server has limited caching storage capacity, it can only cache a subset of services required by IoTDs. Therefore, IoTD k can only offload its task to MEC servers which have cached its required service program. In addition, MEC servers work in a cooperative service-caching manner. That is, if an MEC server cannot provide services to an IoTD, it can offload the subtasks of this IoTD to other adjacent MEC servers or the cloud server caching all services. Let the binary variable $c_{k,n} \in \{0, 1\}$ denote the caching status of service required by IoTD k at MEC server n. Specifically, $c_{k,n} = 1$ means that the service required by IoTD k has been cached by MEC server n; Otherwise $c_{k,n} = 0$.

There are several types of dependencies among subtasks, e.g., sequential dependency, parallel dependency. For IoTD k, we utilize a DAG $G_k \triangleq (V_k, \mathcal{E}_k)$ to describe the dependency relationships among all its subtasks, where $V_k \triangleq \{\nu_{k,i}, \forall i \in \mathcal{I}\}$ is the set of vertices denoting the set of subtasks and $\mathcal{E}_k \triangleq \{e_{k,i,j}, \forall i, j \in \mathcal{I}, i \neq j\}$ is the set of directed edges denoting the dependencies among subtasks of IoTD k. If there exists an edge $e_{k,i,j}$ from $\nu_{k,i}$ to $\nu_{k,j}$, then i is the parent subtask of j and j is the successor subtask of i, where execution of successor subtask j requires the prior completion of its parent subtask i. As shown in Figure 2 [10], one IoTD's computational task undergoes segmentation into 10 subtasks. Therefore, we use 10 network nodes to represent 10 subtasks and directed edges, e.g., $(1, 2)$, to indicate the dependencies among subtasks. We use the binary variable $z^{\mathrm{L}}_{k,i} = \{0, 1\}$ to denote whether subtask i of IoTD k is executed locally, where $z^{\mathrm{L}}_{k,i} = 1$ means that subtask i is executed locally at IoTD k, otherwise $z^{\mathrm{L}}_{k,i} = 0$. Similarly, we use binary variables $z_{k,i,n}$ and $z^{\mathrm{c}}_{k,i}$ to denote whether subtask i of IoTD k is offloaded to MEC server n and the cloud server, respectively, where $= 1$ means offloading and $= 0$ otherwise.

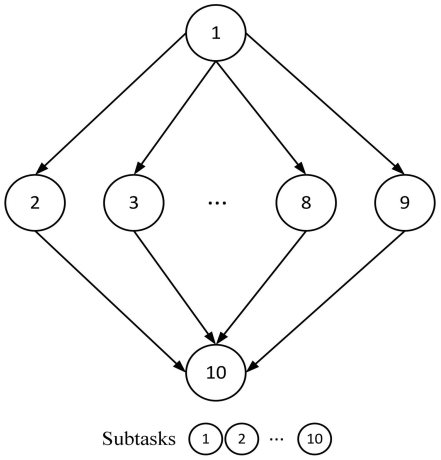

Figure 2. 1-n-1 DAG structure diagram.

2.2. Communication Model

Let $D_{k,i}$ denote the size of input data (including the program codes and input parameters) for subtask i of IoTD k. Also, let $L_{k,i}$ be the computing workload, i.e., the total number of CPU cycles, for processing subtask i of IoTD k, and let f_k^{L} denote the computation resource (number of CPU cycles per second) of IoTD k. Then, the computation execution time of subtask i at IoTD k by local computing is given by:

$$T_{k,i}^{\mathrm{L}} \triangleq \frac{L_{k,i}}{f_k^{\mathrm{L}}}. \tag{1}$$

The energy consumption for subtask i at IoTD k by local computing is given by:

$$E_{k,i}^{\mathrm{L}} \triangleq \varepsilon \left(f_k^{\mathrm{L}}\right)^2 L_{k,i}, \tag{2}$$

where ε represents the switched capacitance, which varies based on the chip's architecture.

If subtask i cannot be processed locally at IoTD k, then it will be first offloaded to IoTD k's nearest MEC server $n_k \in \mathcal{N}$. We denote the transmission time of subtask i to MEC server n_k as T_{k,i,n_k}, which is given by:

$$T_{k,i,n_k} \triangleq \frac{D_{k,i}}{R_{k,n_k}}, \tag{3}$$

where R_{k,n_k} defines the achievable uplink rate for the communication link from IoTD k to MEC server n_k, which is given by:

$$R_{k,n_k} \triangleq W_{n_k,k} \log_2 \left(1 + \frac{P_{k,n_k} \left|H_{k,n_k}\right|^2}{\sum\limits_{\substack{m \in \mathcal{N} \\ m \neq n_k}} \sum\limits_{k' \in \mathcal{K}} P_{k',m} \left|H_{k',n_k}\right|^2 + \sigma^2}\right), \tag{4}$$

where $W_{n_k,k}$ represents the channel bandwidth provisioned by MEC server n_k for IoTD k, P_{k,n_k} and H_{k,n_k} are the uplink transmission power and the channel fading gain from IoTD k to MEC server n_k, and σ^2 is noise power. The energy consumption for transmitting subtask i is given by:

$$E_{k,i,n_k} \triangleq T_{k,i,n_k} P_{k,n_k}. \tag{5}$$

If IoTD k's nearest MEC server n_k has not cached its required service program, or the required service of IoTD k at MEC server n_k has been used, or MEC server n_k does not have enough computation resource, IoTD k's subtask i will be offloaded from MEC server n_k to an adjacent MEC server m which has cached and can provide the service demanded by IoTD k over fiber links. For MEC server n_k, let $\mathcal{M}_{n_k}$ denote the set of MEC servers connecting to it. For subtask i of IoTD k, we can write the transmission time $T_{k,i,m}$ from MEC server n_k to MEC server m as:

$$T_{k,i,m} \triangleq \frac{D_{k,i}}{R_{n_k,m}}, \tag{6}$$

where $R_{n_k,m}$ denotes the transfer rate between MEC server n_k and its adjacent MEC server m over fiber links.

For MEC server n_k, if all MEC servers in $\mathcal{M}_{n_k}$ have not cached the service required by IoTD k, or the service required by IoTD k has been utilized at MEC servers in $\mathcal{M}_{n_k}$, or these MEC servers do not have enough

computation resources, then MEC server n_k will offload subtask i of IoTD k to the cloud center via fiber links. The transmission latency of data with volume of $D_{k,i}$ bits from MEC server n_k to the cloud center is determined by the uplink rate $R_{n_k,\mathrm{c}}$, and then we can formulate the transmission delay as:

$$T_{k,i}^{\mathrm{c}} \triangleq \frac{D_{k,i}}{R_{n_k,\mathrm{c}}}. \tag{7}$$

The subtask i of IoTD k will be executed if it is offloaded to one of the two types of servers which can provide computational service for it. Assuming that the CPU frequency of MEC server n, $\forall n \in \mathcal{N}$, and that of the cloud center allocated to IoTD k are $f_{n,k}$ and $f_{\mathrm{c},k}$, respectively, then we can express the execution time IoTD k's subtask i at MEC server n or the cloud server as follows:

$$T_{k,i}^{\mathrm{exe}} \triangleq \begin{cases} \frac{L_{k,i}}{f_{n,k}}, \text{if subtask } i \text{ of IoTD } k \text{ is processed at MEC server } n, \\ \frac{L_{k,i}}{f_{\mathrm{c},k}}, \text{if subtask } i \text{ of IoTD } k \text{ is processed at the cloud.} \end{cases} \tag{8}$$

In addition, when subtasks i and j are not processed at the same device, e.g., MEC servers, we need to transmit the execution outputs of subtask i to subtask j before executing it. For example, when subtask i is executed locally while subtask j is offloaded to the other two types of servers, i.e., MEC servers and the cloud server, the data results of subtask i are required to be transmitted to subtask j before executing it. We suppose that the data size of the execution results is $u_{k,i,j}$ for IoTD k and subtasks i and j. Then the transmission time for transmitting $u_{k,i,j}$ bits of results from IoTD k to its nearest MEC server n_k is given by:

$$T_{k,i,j}^{\mathrm{off}} \triangleq \frac{u_{k,i,j}}{R_{k,n_k}}, \tag{9}$$

and the transmission energy consumption for transmitting $u_{k,i,j}$ bits of results is:

$$E_{k,i,j}^{\mathrm{off}} \triangleq P_{k,n_k} T_{k,i,j}^{\mathrm{off}}. \tag{10}$$

However, since $u_{k,i,j}$ is in small size and the transmission powers of MEC servers and the cloud server are large enough, we ignore the execution result transmission time between MEC servers, between MEC servers and the cloud, and from MEC servers to IoTD k.

2.3. Computation Model

We utilize $ST_{k,i}$ and $FT_{k,i}$ to represent the starting time and the finishing time of subtask i of IoTD k. $ST_{k,i}$ depends on the following two parts: (1) the transmission time of subtask i for offloading to MEC servers and the cloud server, and (2) the latency incurred until all it's predecessor subtasks' execution outputs are received. We use $T_{k,i}^{\mathrm{tr}}$ to denote the consumed time when subtask i of IoTD k is processed in the following four situations, which is given by:

$$T_{k,i}^{\mathrm{tr}} \triangleq \begin{cases} 0, \text{if subtask } i \text{ of IoTD } k \text{ is processed locally,} \\ T_{k,i,n_k}, \text{if subtask } i \text{ of IoTD } k \text{ is processed at its nearest MEC server } n_k, \\ T_{k,i,n_k} + T_{k,i,m}, \text{if subtask } i \text{ of IoTD } k \text{ is processed at MEC server } m \in \mathcal{M}_{n_k}, \\ T_{k,i,n_k} + T_{k,i}^{\mathrm{c}}, \text{if subtask } i \text{ of IoTD } k \text{ is processed at the cloud.} \end{cases} \tag{11}$$

The first case corresponds to the scenario when subtask i is processed locally at IoTD k. The second case corresponds to the scenario when subtask i is offloaded to and processed at IoTD k's nearest MEC server n_k. The third case corresponds to the case when offloading subtask i to the server adjacent to MEC server n_k. The last case corresponds to the case when subtask i is executed in the cloud.

Let $T_{k,i}^{\mathrm{re}}$ be the time when subtask i of IoTD k receives the execution outputs of all its predecessor subtasks, which is formulated as:

$$T_{k,i}^{\mathrm{re}} \triangleq \max_{i' \in \mathrm{pred}(i)} \left\{ FT_{k,i'} + T_{k,i',i}^{\mathrm{off}} \right\}, \tag{12}$$

where pred (i) denotes the set of the predecessor subtasks for subtask i, $T_{k,i',i}^{\mathrm{off}}$ denotes the transmission time of data results from all predecessor subtasks of IoTD k's subtask i to subtask i, and $FT_{k,i'}$ is the finishing time of subtask i'. Therefore, we can write the starting time of subtask i of IoTD k as:

$$ST_{k,i} \triangleq \max\left\{\sum_{i' \in \text{pred}(i)}^{i} T^{\text{tr}}_{k,i'}, T^{\text{re}}_{k,i}\right\}, \tag{13}$$

where $\sum_{i' \in \text{pred}(i)}^{i} T^{\text{tr}}_{k,i'}$ is the sum of $T^{\text{tr}}_{k,i}$ of all the predecessor subtasks i' of IoTD k's subtask i.

The start time $ST_{k,i}$ of a subtask is critically dependent on the completion of all its predecessors due to data dependencies defined by the DAG. As formulated in Equation (12), the time $T^{\text{re}}_{k,i}$ represents the moment when all necessary input data from predecessor subtasks have been received. Specifically, for a given subtask i, the system cannot process this subtask until every predecessor subtask $i' \in \text{pred}(i)$ has been finished. The waiting time is determined by the slowest predecessor, i.e., the one with the largest sum of finishing time $FT_{k,i'}$ and result transmission time $T^{\text{off}}_{k,i',i}$. This is captured by the max operator in Equation (12), which ensures that the subtask only becomes ready when its most delayed predecessor's data have arrived. Consequently, the start time $ST_{k,i}$ in Equation (13) is defined as the maximum value between (a) the time taken to offload the subtask itself to the server, i.e., $T^{\text{tr}}_{k,i}$, and (b) the time required to gather all predecessor results, i.e., $T^{\text{re}}_{k,i}$. This formulation guarantees that the dependency constraints are strictly satisfied before execution begins.

Therefore, the finishing time for subtask i is formulated as:

$$FT_{k,i} \triangleq \begin{cases} ST_{k,i} + T^{\text{L}}_{k,i}, \text{if subtask } i \text{ of IoTD } k \text{ is processed locally,} \\ ST_{k,i} + T^{\text{exe}}_{k,i}, \text{otherwise.} \end{cases} \tag{14}$$

The energy consumption at IoTD k for transmitting subtask i and the results of its predecessors is given by:

$$E_{k,i} \triangleq E_{k,i,n_k} + \sum_{i' \in \text{pred}(i)} E^{\text{off}}_{k,i',i}, \tag{15}$$

where $\sum_{i' \in \text{pred}(i)} E^{\text{off}}_{k,i',i}$ represents the energy consumption at IoTD k for transmitting the computation results of subtasks in the set $\text{pred}(i)$ to subtask i. Applying the preceding equations yields the following maximum finishing time for all IoTDs:

$$T \triangleq \max_{k \in \mathcal{K}} \{FT_{k,I}\}. \tag{16}$$

Meanwhile, we can derive the total energy consumption at IoTDs as follows:

$$E \triangleq \sum_{k \in \mathcal{K}}^{K} \sum_{i \in \mathcal{I}}^{I} \left(z^{\text{L}}_{k,i} E^{\text{L}}_{k,i} + \left(\sum_{n \in \mathcal{N}} z_{k,i,n} + z^{\text{c}}_{k,i} \right) E_{k,i} \right). \tag{17}$$

2.4. The Optimization Problems Formulations

Similar to [16], our objective is to minimize ETC, of all IoTDs for task processing subject to the offloading decisions of all subtasks, the computation resources and service-caching of all MEC servers, and energy consumption of each IoTD. Thus, the ETC minimization problem is derived as:

$$\min_{\mathcal{C},\mathcal{F},\mathcal{Z}} \chi \triangleq \lambda T + (1-\lambda)E, \tag{18}$$

s.t.:

$$\textbf{C1} : \sum_{k \in \mathcal{K}} (c_{k,n}\beta_k) \leq C_n, \forall n \in \mathcal{N},$$

$$\textbf{C2} : c_{k,n} \leq 1, \forall n \in \mathcal{N}, \forall k \in \mathcal{K},$$

$$\textbf{C3} : z^{\text{L}}_{k,i} + \sum_{n \in \mathcal{N}} z_{k,i,n} + z^{\text{c}}_{k,i} = 1, \forall k \in \mathcal{K},$$

$$\textbf{C4} : \sum_{i \in \mathcal{I}}^{I} \left(z^{\text{L}}_{k,i} E^{\text{L}}_{k,i} + \left(\sum_{n \in \mathcal{N}} z_{k,i,n} + z^{\text{c}}_{k,i} \right) E_{k,i} \right) \leq E_k, \forall k \in \mathcal{K},$$

$$\textbf{C5} : \sum_{k \in \mathcal{K}} f_{n,k} \leq f^{\max}_n, \forall n \in \mathcal{N},$$

where

$$\begin{cases} \mathcal{C} \triangleq \{c_{k,n} \in \{0,1\}, \forall n \in \mathcal{N}, \forall k \in \mathcal{K}\}, \\ \mathcal{F} \triangleq \{0 \leq f_{n,k} \leq f_n^{\max}, \forall n \in \mathcal{N}, \forall k \in \mathcal{K}\}, \\ \mathcal{Z} \triangleq \left\{z_{k,i}^{\mathrm{L}}, z_{k,i,n}, z_{k,i}^{\mathrm{c}} \in \{0,1\}, \forall n \in \mathcal{N}, \forall k \in \mathcal{K}\right\}, \end{cases}$$

where $\lambda \in [0,1]$ represents the weight trading off delay and energy consumption, T and E represent time and energy consumption, which are given by Equations (16) and (17), respectively, C_n is MEC server n's maximum caching storage capacity, β_k is the data size of IoTD k's required service program, and $f_n^{\max}$ is MEC server n's maximum computation resource. Besides, **C1** is the caching capacity constraint of each MEC server. **C3** ensures that each subtask can only be executed at IoTDs or be offloaded to an MEC server or the cloud. **C4** is the energy consumption constraint of IoTD k. **C5** guarantees that for MEC server n the aggregate of computation resources allocated to all IoTDs is bounded by its maximum computation resource.

However, it is highly challenging to solve the optimization problem in Equation (18) due to its mixed-integer nonlinear programming (MINLP) nature, which makes this problem NP-hard. The complexity arises from three main factors: (i) the combinatorial explosion of the solution space caused by the joint optimization of binary caching variables $c_{k,n}$'s, binary offloading decisions, i.e., $z_{k,i}^{\mathrm{L}}$'s, $z_{k,i,n}$'s, $z_{k,i}^{\mathrm{c}}$'s, and continuous resource allocation variables $f_{n,k}$'s; (ii) the non-convex constraints, e.g., **C3**; and (iii) the DAG-based task dependencies that propagate the impact of each decision across subtasks. Therefore, finding the global optimal solution in polynomial time is intractable, motivating the development of an efficient and low-complexity DRL-based approach.

3. GCN and DRL Based Dependency-Aware Computation-Offloading and Cooperative Service-Caching Scheme

As shown in Figure 3, our proposed framework consists of three main modules: GCN Embedding Module: Processing each IoTD's DAG to generate dependency-aware subtask embeddings. State Construction Module: Combining the GCN embeddings with environmental features (caching status, channel gains, resource-allocation, etc.) to form a unified state representation. A2C Decision Module: Taking the state as input and outputting joint caching, resource-allocation, and offloading decisions via actor-critic learning. In this part, we propose a method that uses GCN to assist DRL for solving the optimization problem defined by Equation (18).

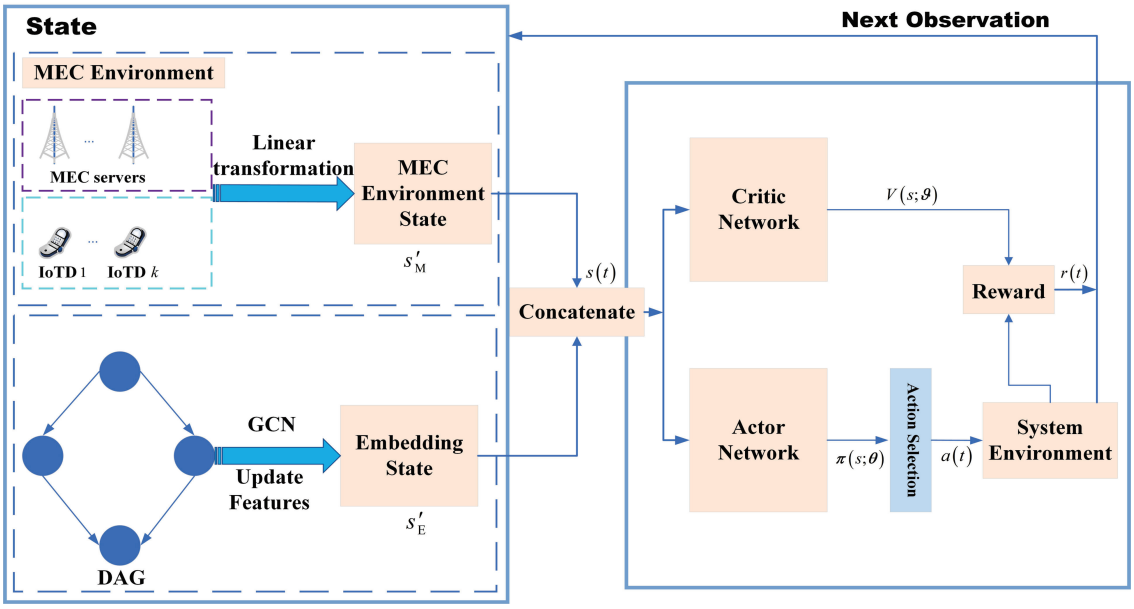

Figure 3. DRL-based MEC computation-offloading framework.

3.1. DRL-Based Framework

Complex optimization problems can be addressed by DRL. In general, DRL is particularly effective for solving complex optimization problems that are non-convex, large-scale, and involve both discrete and continuous decision variables, where traditional optimization methods often fail to find optimal solutions within polynomial time. Such problems are prevalent in MEC due to the dynamic nature of wireless channels, task dependencies, and resource constraints. By interacting with the environment, DRL agents can learn near optimal policies without requiring

explicit model knowledge, making them suitable for the MINLP problem formulated in this paper. However, it is challenging to address the optimization problem defined by Equation (18) by only using DRL when there exist dependencies among subtasks. Specifically, it is not easy to model the dependencies among subtasks, since DRL ignores the dependencies of tasks/subtasks. But, GCN can extract the features of subtasks from the DAG and represent their dependencies by using an adjacency matrix. Moreover, by using GCN, we can update the original features of subtasks, based on which we can employ DRL to make offloading decisions for these subtasks. Therefore, similar to [10, 18], we utilize the GCN-aided DRL method to tackle the dependency-aware subtask offloading problem.

The framework of GCN-aided DRL is displayed in Figure 3. First, the state of DRL includes the MEC environment state s'_{M} and the embedding state s'_{E}, which are obtained by the linear transformation of the environment feature matrix s_{M} and the embedding features s_{E}, respectively. (We give an example to explain the linear transformation. Suppose $\mathbf{A}_1$ is a 5×6 matrix and $\mathbf{A}_2$ is a 3×4 matrix. Since their dimensions do not match, they cannot be directly concatenated. Therefore, we need to transform the number of their rows into the same number, i.e., $\max\{5, 3\}$. Each element of the two added rows of matrix $\mathbf{A}_2$ can be equal to zero. In this way, $\mathbf{A}_2$ becomes a 5×4 matrix. Then, both $\mathbf{A}_1$ and $\mathbf{A}_2$ have the same number of rows, so that they can be concatenated horizontally to form a 5×10 matrix.) The representation of the embedding feature matrix s_{E} is as follows:

$$s_{\mathrm{E}} \triangleq \mathrm{GCN}\left\{X_k, \mathrm{adj}(G_k)\right\}, \tag{19}$$

where X_k is a $I \times 2$ subtask feature matrix, where the first and second columns of X_k represent the required CPU cycles and data sizes of I subtasks of IoTD k, respectively. For IoTD k's DAG G_k with I nodes, its adjacency matrix, denoted by $\mathrm{adj}(G_k)$, is a $I \times I$ matrix representing the dependency relationship among subtasks. If there exists an edge (i, j), then $[\mathrm{adj}(G_k)]_{ij} = 1$; Otherwise, $[\mathrm{adj}(G_k)]_{ij} = 0$. Meanwhile, $\mathrm{GCN}\left\{X_k, \mathrm{adj}(G_k)\right\}$ indicates that we extract X_k from IoTD k's DAG G_k and $\mathrm{adj}(G_k)$, and update the original features X_k with GCN into a new $I \times 2$ matrix, i.e., embedding feature s_{E}.

The MEC environment feature matrix s_{M} is defined as follows:

$$s_{\mathrm{M}} \triangleq \mathrm{concat}\left\{\mathbf{c}, \mathbf{F}, \mathbf{H}, \mathbf{Z}_{\mathrm{L}}, \mathbf{Z}, \mathbf{Z}_{\mathrm{c}}\right\}, \tag{20}$$

where $\mathbf{c} \triangleq [C_1, \ldots, C_n, \ldots, C_N]$ denotes the caching capacity vector of MEC servers. Moreover, $\mathbf{F} \triangleq [\mathbf{f}_1, \ldots, \mathbf{f}_n, \ldots, \mathbf{f}_N] \in \mathbb{R}^{K \times N}$ denotes the computation resource-allocation matrix of MEC server n and $\mathbf{H} \triangleq [\mathbf{h}_1, \ldots, \mathbf{h}_n, \ldots, \mathbf{h}_N] \in \mathbb{R}^{K \times N}$ denotes channel gain matrix, where $\mathbf{f}_n, \mathbf{h}_n \in \mathbb{R}^{K \times 1}$ are respectively the nth column vectors of $\mathbf{F}$ and $\mathbf{H}$. In addition, $\mathbf{Z} \triangleq [\mathbf{Z}_1, \ldots, \mathbf{Z}_n, \ldots, \mathbf{Z}_N] \in \mathbb{R}^{K \times NI}$, where $\mathbf{Z}_n \triangleq [\mathbf{z}_{1,n}, \ldots, \mathbf{z}_{i,n}, \ldots, \mathbf{z}_{I,n}]$ is the offloading strategy matrix for whether subtasks are executed on MEC server n, and $\mathbf{Z}_{\mathrm{L}} \triangleq [\mathbf{z}_1^{\mathrm{L}}, \ldots, \mathbf{z}_i^{\mathrm{L}}, \ldots, \mathbf{z}_I^{\mathrm{L}}] \in \mathbb{R}^{K \times I}$ and $\mathbf{Z}_{\mathrm{c}} \triangleq [\mathbf{z}_1^{\mathrm{c}}, \ldots, \mathbf{z}_i^{\mathrm{c}}, \ldots, \mathbf{z}_I^{\mathrm{c}}] \in \mathbb{R}^{K \times I}$ respectively represent the offloading strategy matrices for whether subtasks are executed at IoTDs or in the cloud, where $\mathbf{z}_i^{\mathrm{L}}, \mathbf{z}_{i,n}, \mathbf{z}_i^{\mathrm{c}} \in \mathbb{R}^{K \times 1}$ are respectively the ith column vectors of $\mathbf{Z}_{\mathrm{L}}$, $\mathbf{Z}_n$, and $\mathbf{Z}_{\mathrm{c}}$. Besides, the term concat $\{\mathrm{X}, \mathrm{Y}\}$ means concatenating all the elements inside into a matrix. Since dimensions of the above elements are inconsistent, the above elements are linearly transformed into the same dimension and then concatenated to obtain the MEC environment feature matrix s_{M}.

3.2. State

From the above, it can be known the system state includes MEC environment state s'_{M} and embedding state s'_{E}. Then, we can define the state as follows:

$$s(t) \triangleq \mathrm{concat}\left\{s'_{\mathrm{M}}(t), s'_{\mathrm{E}}(t)\right\}. \tag{21}$$

By using DRL, the actor network outputs action $a(t)$ for state $s(t)$, triggering immediate reward $r(t)$ and state transitioning to $s(t+1)$ through environmental interaction. Then, we can construct the Markov Decision Process (MDP).

The procedure of state embedding and obtaining the state is shown in Algorithm 1.

Algorithm 1 State Embedding Algorithm.

Input: $G_k \triangleq (V_k, \mathcal{E}_k)$, adjacency matrix adj$(G_k)$, subtask feature matrix X_k and MEC environment feature matrix s_{M}.
output: DRL state.
For each GCN layer $= 1, 2, \ldots,$ **do**,
  Reset the environment.
  Apply GCN on X_k and adj(G_k) to obtain s_{E}.
  Apply linear transformation on s_{M} to obtain s'_{M}.
  Apply linear transformation on s_{E} to obtain s'_{E}.
  Concatenate s'_{M} and s'_{E} to yield state s for DRL.
End for
Output s

3.3. Action

We denote the action at time step t as $a(t)$. The agent decides the types of services needed to be cached by MEC servers $c_{k,n}$, the CPU frequency $f_{n,k}$ of MEC server n allocated to IoTD k, $\forall k \in \mathcal{K}$, $\forall n \in \mathcal{N}$, and the offloading decisions $z^{\mathrm{L}}_{k,i}$, $z_{k,i,n}$, and $z^{\mathrm{c}}_{k,i}$ for all subtasks of IoTDs, in light of the current state $s(t)$ observed. Hence, we can define the action function $a(t)$ as follows:

$$a(t) \triangleq \left\{ \{c_{k,n}(t)\}_{k\in\mathcal{K},n\in\mathcal{N}}, \{f_{n,k}\}_{k\in\mathcal{K},n\in\mathcal{N}}, \left\{z^{\mathrm{L}}_{k,i}, z_{k,i,n}, z^{\mathrm{c}}_{k,i}\right\}_{k\in\mathcal{K},i\in\mathcal{I},n\in\mathcal{N}} \right\}. \tag{22}$$

3.4. Reward

The reward function is formulated as the negative form of ETC, motivated by the minimization objective for all IoTDs in Equation (18). In addition, to maximize the reward, subtasks should be preferentially executed either at IoTDs or on MEC servers. Consequently, the reward function at time step t is formulated as:

$$r(t) \triangleq -\chi(t) + \Upsilon_1(t) + \Upsilon_2(t) + \Upsilon_3(t), \tag{23}$$

where $\Upsilon_1(t)$, $\Upsilon_2(t)$, and $\Upsilon_3(t)$ are given by:

$$\Upsilon_1(t) \triangleq \begin{cases} \phi, & z^{\mathrm{L}}_{k,i} = 1, \quad z_{k,i,n} = z^{\mathrm{c}}_{k,i} = 0, \\ \phi, & z_{k,i,n} = 1, z^{\mathrm{L}}_{k,i} = z^{\mathrm{c}}_{k,i} = 0, \\ \varphi_{k,i}, & z^{\mathrm{c}}_{k,i} = 1, \quad z_{k,i,n} = z^{\mathrm{L}}_{k,i} = 0, \end{cases} \tag{24}$$

$$\Upsilon_2(t) \triangleq \begin{cases} \varphi^{\mathrm{en}}_k, \sum_{i=1}^{I} \left(E^{\mathrm{L}}_{k,i} + E_{k,i,k_n} \right) > E_k, \\ 0, \quad \text{otherwise}, \end{cases} \tag{25}$$

$$\Upsilon_3(t) \triangleq \begin{cases} \varphi^{\mathrm{fre}}_k, \sum_{k\in\mathcal{K}} f_{n,k} > f^{\max}_n, \\ 0, \quad \text{otherwise}, \end{cases} \tag{26}$$

where $\varphi_{k,i}$, φ^{en}_k, and φ^{fre}_k are all negative constants and ϕ is a positive constant. Equation (24) specifies that executing subtasks locally or on MEC servers yields an intermediate reward ϕ, while offloading to the cloud incurs a penalty $\varphi_{k,i}$. In addition, penalties φ^{en}_k and φ^{fre}_k are imposed to account for the energy constraints of IoTDs and computation resources limitations of MEC servers **C4** and **C5**, respectively.

3.5. Actor-Critic Framework

To address optimization problems involving both discrete and continuous variables, our actor network employs a multi-head output architecture. For discrete actions (e.g., caching decisions $c_{k,n}$ and offloading decisions $z^{\mathrm{L}}_{k,i}$, $z_{k,i,n}$, $z^{\mathrm{c}}_{k,i}$), we use softmax output heads that produce probability distributions from which actions are sampled. For continuous actions (e.g., resource-allocation $f_{n,k}$), the actor outputs the mean of a Gaussian distribution to generate continuous values. These outputs are produced in a single forward pass, allowing the network to capture interdependencies between action types. The critic network then takes the concatenated state and action vector as input to estimate the state-value or action-value function. During training, the policy gradient is computed using the joint log-probability of the sampled hybrid action, while an entropy term encourages exploration across both

discrete and continuous spaces. This design enables our A2C framework to efficiently solve the mixed-integer nonlinear programming problem in Equation (18).

In this paper, we implement the advantage actor-critic (A2C) algorithm, a method grounded in the actor-critic framework, to determine action selection $a(t)$ at each time step t. The A2C algorithm utilizes two deep neural networks (DNNs): (1) an actor network generating a policy function $\pi\left(s;\boldsymbol{\theta}\right)$ that outputs a probability distribution over feasible actions given state $s(t)$, where $\boldsymbol{\theta}$ denotes the network's parameter vector. The action $a(t)$ is sampled from this distribution and is executed in the current state $s(t)$; (2) a critic network evaluating the value function of the current state to guide policy updates. Upon action execution, the environment transitions to state $s(t+1)$ and the agent receives an immediate reward $r(t)$. Furthermore, the critic network computes a state-value function $V\left(s;\boldsymbol{\vartheta}\right)$ to assess the value of the current state $s(t)$, where $\boldsymbol{\vartheta}$ is the critic network's parameter vector. In addition, to enhance training stability and efficiency, A2C generates an advantage function $A\left(s,a;\boldsymbol{\vartheta}\right)$, which quantifies the relative benefit of selecting action a in state s compared to the average behavior of the policy. Following prior works [33,34], the advantage function can be expressed as:

$$A\left(s,a;\boldsymbol{\vartheta}\right)\triangleq Q\left(s,a;\boldsymbol{\vartheta}\right)-V\left(s;\boldsymbol{\vartheta}\right), \tag{27}$$

where $Q\left(s,a;\boldsymbol{\vartheta}\right)$ is the Q-function of DRL that is used to evaluate action a selected in state s. Nevertheless, for updating the parameter vectors $\boldsymbol{\vartheta}$ and $\boldsymbol{\theta}$, the critic network must estimate the Q-function $Q\left(s,a;\boldsymbol{\vartheta}\right)$ and the state-value function $V\left(s;\boldsymbol{\vartheta}\right)$. The A2C framework can effectively approximate $V\left(s;\boldsymbol{\vartheta}\right)$, enabling the estimation of the Q-value $Q\left(s,a;\boldsymbol{\vartheta}\right)$ at time step t based on the following formula:

$$Q\left(s,a;\boldsymbol{\vartheta}\right)\approx\bar{r}(t)+\gamma V\left(s(t+1);\boldsymbol{\vartheta}\right), \tag{28}$$

where $\bar{r}(t)$ denotes the average reward received at time step t, which is formulated as:

$$\bar{r}(t)\triangleq\frac{\sum_{t'=1}^{t}r(t')}{t}. \tag{29}$$

To ensure reward convergence, we adopt the average reward $\bar{r}(t)$ instead of the instant reward $r(t)$. Subsequently, to address the inherent bias in critic's value estimation, the A2C algorithm estimates the advantage function's value $A\left(s,a;\boldsymbol{\vartheta}\right)$ at time step t based on the temporal difference (TD) error as follows:

$$\delta\left(s(t),\boldsymbol{\vartheta}\right)\triangleq\bar{r}(t)+\gamma V\left(s(t+1);\boldsymbol{\vartheta}\right)-V\left(s(t);\boldsymbol{\vartheta}\right), \tag{30}$$

where $\gamma\in(0,1]$, representing the discount factor, defines the weight of future rewards. To update the parameter vector $\boldsymbol{\theta}$ of the actor network, the A2C algorithm minimizes its loss function $L\left(\boldsymbol{\theta}\right)$, which is given by:

$$L\left(\boldsymbol{\theta}\right)\triangleq-[\log\pi\left(s;\boldsymbol{\theta}\right)\delta\left(s(t),\boldsymbol{\vartheta}\right)+\beta H\left(\pi\left(s;\boldsymbol{\theta}\right)\right)], \tag{31}$$

where $\beta\in(0,1)$ is a weight parameter and $H\left(\pi\left(s;\boldsymbol{\theta}\right)\right)$ serves as the entropy function, instrumental in promoting the exploration of environments. Following the same approach, the A2C algorithm improves the critic network's parameters $\boldsymbol{\vartheta}$ by minimizing this loss function:

$$L\left(\boldsymbol{\vartheta}\right)\triangleq\left[\delta\left(s(t),\boldsymbol{\vartheta}\right)\right]^{2}=[\bar{r}(t)+\gamma V\left(s(t+1);\boldsymbol{\vartheta}\right)-V\left(s(t);\boldsymbol{\vartheta}\right)^{2}]. \tag{32}$$

The A2C algorithm subsequently updates the actor network's parameter $\boldsymbol{\theta}$ and critic network's parameter $\boldsymbol{\vartheta}$ according to the following update rules:

$$\boldsymbol{\theta}\leftarrow\boldsymbol{\theta}+\alpha\left[\delta\left(s(t),\boldsymbol{\vartheta}\right)\frac{\nabla\log\pi\left(a(t)|s(t);\boldsymbol{\theta}\right)}{\nabla\boldsymbol{\theta}}+\beta\frac{\nabla H\left(\pi\left(s;\boldsymbol{\theta}\right)\right)}{\nabla\boldsymbol{\theta}}\right], \tag{33}$$

$$\boldsymbol{\vartheta}\leftarrow\boldsymbol{\vartheta}+\widetilde{\alpha}\delta\left(s(t),\boldsymbol{\vartheta}\right)\frac{\nabla V\left(s(t);\boldsymbol{\vartheta}\right)}{\nabla\boldsymbol{\vartheta}}, \tag{34}$$

where $\alpha\in(0,1)$ and $\widetilde{\alpha}\in(0,1)$ denote the learning rates for the actor and critic networks, respectively.

3.6. Computational Complexity Analyses

Algorithm 2 outlines our A2C-based scheme for solving the optimization problem in Equation (18). Let L and $\widetilde{L}$ denote the numbers of hidden layers of each actor and critic networks, respectively, and Z_{h} and $\widetilde{Z}_{\mathrm{h}}$ be the

numbers of neurons in the hth layer of the corresponding neural networks, respectively. In Algorithm 2, there is one actor network and one critic network, both of which are fully connected. According to [35], the total computational complexity of Algorithm 2 is $O(\Gamma(\iota/\upsilon)(\tau Z_1 + \sum_{\mathrm{h}=2}^{L} Z_{\mathrm{h}-1} Z_{\mathrm{h}} + 2\rho Z_L + \tau \widetilde{Z}_1 + \sum_{\mathrm{h}=2}^{\widetilde{L}} \widetilde{Z}_{\mathrm{h}-1} \widetilde{Z}_{\mathrm{h}} + \widetilde{Z}_{\widetilde{L}}))$, where ι is the max training steps of each episode, Γ is the number of episodes, υ is the sampling interval, and τ and ρ are the numbers of neurons in the input and output layers of each actor or critic network, respectively.

Algorithm 2 A2C-Based Scheme for Solving the Optimization Problem in Equation (18).

1: **Initialize**: Actor and critic networks' parameters $\boldsymbol{\theta}$ and $\boldsymbol{\vartheta}$, discount factor γ, actor learning rate α, and critic learning rate $\widetilde{\alpha}$.
2: **For** each episode $= 1, 2, \ldots, \Gamma$ **do**,
3: Reset the environment.
4: **For** each time step $t = 1, 2, \ldots, \iota$ **do**,
5: Obtain state $s(t)$ by using Algorithm 1.
6: The actor network derives the action probability distribution for a given state $s(t)$ through the policy function $\pi(s; \boldsymbol{\theta})$.
7: Utilizing the value function $V(s; \boldsymbol{\vartheta})$, the critic network assesses the value of state $s(t)$.
8: The agent stochastically selects action $a(t)$ according to the derived action probability distribution in line 6.
9: The agent receives reward $r(t)$, updates the average reward $\bar{r}(t)$, and transitions to next state $s(t+1)$.
10: The critic network evaluates the TD error using Equation (30) for the given current state $s(t)$, the executed action $a(t)$, the obtained average $\bar{r}(t)$, and the next state $s(t+1)$.
11: Update $\boldsymbol{\theta}$ of the critic network by using Equation (33).
12: Update $\boldsymbol{\vartheta}$ of the actor network by using Equation (34).
13: **End for**
14: **End for**

4. Performances Evaluations

MEC servers and IoTDs are deployed within a 50 m $\times$ 50 m area, and the fading between IoTDs and MEC servers follows Rayleigh fading. Unless otherwise stated, for each IoTD k, we take $f_k^{\mathrm{L}} = 10^9$ Hz and the penalty parameters $\varphi_k^{\mathrm{en}} = -0.2$ in Equation (25). Besides, we take $f_n^{\max} = 10^{10}$ Hz for MEC server n, $\forall n \in \mathcal{N}$, and the penalty parameter $\varphi_k^{\mathrm{fre}} = -0.2$ in Equation (26). We take positive reward $\phi = 0.2$ and the penalty parameter $\varphi_{k,i} = -0.2$ in Equation (24). Moreover, we take the noise power as $\sigma^2 = 10^{-9}$ W, the effective switched capacitance as $\varepsilon = 10^{-28}$, the weight parameter $\lambda = 0.5$ in Equation (18), the CPU frequency of the cloud center allocated to IoTD k as $f_{\mathrm{c},k} = 10^9$ Hz, and the channel bandwidth allocated to IoTD k by its nearest MEC server n_k as $W_{n_k,k} = 20$ MHz. In addition, for all users, the data input size, the total number of CPU cycles, and the data result of subtask i follow uniform distribution with $L_{k,i} \in [1 \times 10^5, 2 \times 10^5]$ cycles, $D_{k,i} \in [300, 500]$ KB, and $u_{k,i,j} \in [10, 100]$ KB, respectively. We also set the number of MEC servers as $N = 4$ and the number of IoTDs as $K = 12$ each with $I = 10$ subtasks. The specific values of the hyperparameters are shown in Table 3.

Table 3. Hyperparameter.

Hyperparameter	Description	Value
$\boldsymbol{\vartheta}$	Actor learning rate	0.00001
$\boldsymbol{\theta}$	Critic learning rate	0.0001
Γ	Number of episodes	300
ι	Steps per episode	1000
υ	Soft update factor	2
β	Weight of entropy	0.01
γ	Discount factor	0.99

To evaluate the performance of our proposed scheme Algorithm 2, we also consider the following baseline schemes:

- *Actor-Critic (AC)*: This scheme has an actor network and a critic network and it is capable of solving our optimization problem with both discrete and continuous variables.

- *Proximal Policy Optimization (PPO)*: We use the PPO-clip algorithm to solve our optimization problem given in Equation (18) which has both discrete and continuous variables.
- *Deep Deterministic Policy Gradient combined with Double Deep Q Network (DDPG+DDQN)*: DDPG is an actor-critic algorithm designed for continuous action spaces. It maintains an actor network that deterministically maps states to continuous actions, and two critic networks (a primary Q network and a target Q network) that evaluate the action-value function. In our implementation, DDPG is responsible for outputting the continuous resource allocation variables $f_{n,k}$. DDQN has a Q network and a target Q network (instead of the DQN that overestimates the Q-value) to decide discrete caching $c_{n,k}$ and offloading decisions $z^{\mathrm{L}}_{k,i}, z_{k,i,n}, z^{\mathrm{c}}_{k,i}$. The two algorithms operate in a coordinated manner: at each time step, the agent observes the current state $s(t)$, and then DDPG and DDQN simultaneously generate the continuous and discrete action components, respectively. These components are combined to form the complete action $a(t)$, which is then executed in the environment. The shared reward $r(t)$ is used to update both algorithms, allowing them to learn actions cooperatively.
- *Soft Actor-Critic combined with Dueling DQN (SAC+DDQN)*: SAC has one actor network and four critic networks (two primary Q networks and two target Q networks). The core of SAC's design lies in enhancing action exploration through entropy maximization and improving stability by combining two critic networks and soft update of target network parameters, and we use SAC to determine continuous variables. Entropy maximization for action exploration: Entropy measures the randomness of the policy. Maximizing entropy encourages the policy to assign higher probabilities to a diverse range of actions rather than converging prematurely to a deterministic policy. This mechanism promotes extensive exploration of the action space, preventing the agent from getting trapped in suboptimal local optima. Dual critic networks for stability improvement: SAC maintains two independent Q networks and their corresponding target networks. During training, both Q networks estimate the action-value function, and the minimum of the two estimates is used to compute the target value. This technique, known as "sliced double Q", effectively mitigates the overestimation bias commonly found in value-based methods. Soft target network updates for smooth learning: At each training step, the target network parameters are gradually interpolated with the current network parameters by using a small update coefficient. This progressive update ensures a smoother and more stable transition of target values, reducing the risk of drastic policy fluctuations and enhancing convergence stability. Dueling DQN has a Q network and a target Q network and we use it to determine discrete variables.

We adopt a sequential chain-structured DAG as shown in Figure 4 [10], where each node (except the first and last nodes) has exactly one predecessor and one successor, with no shared predecessors or successors among nodes. In addition, we set the transmission rate between two adjacent MEC servers n and m as $R_{n,m} = 7$ Mbps and the transmission rate from MEC server n to the cloud as $R_{n,\mathrm{c}} = 7$ Mbps. The first stage of our analysis compares the convergence performances of Algorithm 2 against the four baseline schemes. Analyzing Figure 5, we can observe that Algorithm 2 achieves the highest average reward, converging within approximately 70 episodes. Experimental results reveal that the AC algorithm yields significantly larger rewards relative to the other baseline schemes and has stable convergence performance. However, it requires approximately 150 episodes to reach convergence, and its final convergence reward is lower than that of Algorithm 2. This phenomenon may stem from two key limitations of AC: (1) inherent bias in the critic's value estimation and (2) instability in policy gradient updates due to high variance. Algorithm 2 incorporates an advantage function, which reduces the variance of policy gradient estimates by using the state value as a baseline to evaluate the relative advantage of each action, thereby stabilizing training and accelerating convergence. The actor-critic co-update mechanism mitigates policy oscillation risks, making it particularly suitable for complex control tasks. What's more, the rewards of the PPO algorithm increase from the beginning to the 25th episode and then fluctuate slightly until the 50th episode. This is because PPO needs to maintain prolonged exploration while avoiding premature convergence to suboptimal local solutions. The DDPG+DDQN algorithm achieves convergence at approximately 70 episodes, exhibiting the lowest reward. This is because both DDPG and DDQN are stably trained using the target networks. However, if the update frequency (e.g., the soft update parameter of DDPG and the update interval of DDQN's target network) is not set properly, it may lead to Q-value estimation deviation or lag which may cause reward fluctuations. Furthermore, the SAC+DDQN algorithm fluctuates greatly after convergence, resulting in a significant increase in training time and computer memory. This is because SAC needs to maintain five DNNs and the implementation complexity of DDQN is high, which lead to unstable training performance.

To assess how the DAG structure affects Algorithm 2's performance relative to the baseline schemes, we set the DAG structure as 1-n-1 as shown in Figure 2, where subtask 1 and subtask 10 are the first and last subtasks, respectively, and n represents the parallel execution of subtasks 2-9 with no dependencies among them. Therefore, the completion time of parallel subtasks equals the maximum execution time among these subtasks. The completion

time of all subtasks of an IoTD is the sum of the execution time of the start subtask and the end subtask, plus the maximum transmission time of the results of the parallel subtasks and the completion time of the parallel subtasks. The comparative performance evaluation between Algorithm 2 and the four baseline scheme under a 1-n-1 DAG structure is illustrated in Figure 6. Analyzing Figure 6, it is evident that Algorithm 2 still has the highest reward and the most stable convergence performance, indicating the reliability of Algorithm 2. Although the convergence performance of the AC algorithm is stable, its reward is relatively slow. Moreover, PPO also has fluctuations at the beginning, and then the rewards increase to converge. This is still because PPO needs to maintain prolonged exploration while avoiding premature convergence to suboptimal local solutions. In addition, DDPG+DDQN and SAC+DDQN have large fluctuations and low convergence rewards, indicating that they are not suitable for solving our formulated optimization problem.

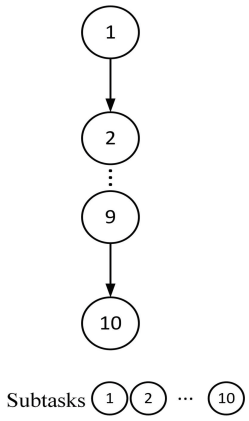

Figure 4. Sequential chain-structured DAG.

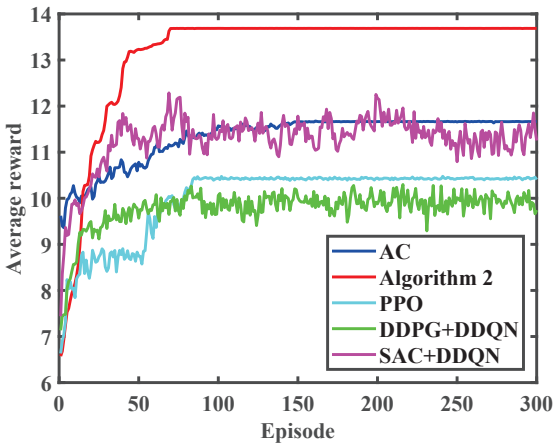

Figure 5. Convergence performance comparison between Algorithm 2 and baseline schemes for sequential chain-structured DAG.

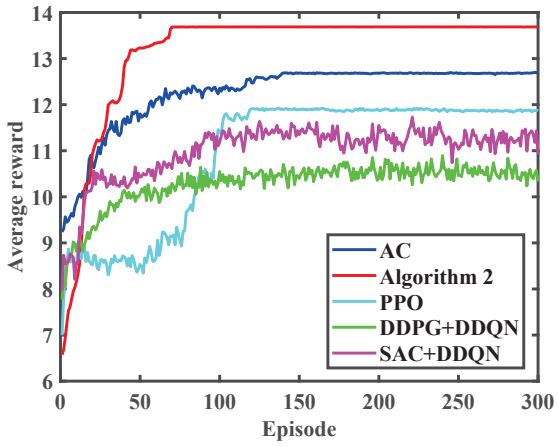

Figure 6. Convergence performance comparison between Algorithm 2 and baseline schemes for 1-n-1 DAG.

In Figure 7, we show the performance comparison of Algorithm 2 with the four baseline schemes under different transmission rates $R_{n,m}$ between MEC servers. As illustrated in Figure 7a, when the transmission rate $R_{n,m}$ between MEC servers increases incrementally by 2 Mbps (from $R_{n,m} = 4.5$ Mbps to $R_{n,m} = 12.5$ Mbps), the total energy consumption of all IoTDs for all the five algorithms gradually decreases as a whole. Although Algorithm 2 exhibits marginally higher energy consumption than some baseline schemes at $R_{n,m} = 6.5$ Mbps and $R_{n,m} = 10.5$ Mbps, it achieves the lowest energy consumption in most cases, demonstrating superior adaptability to dynamic environments. Figure 7b shows the maximum task completion time for all IoTDs. As expected, the maximum task completion time generally does not decrease as $R_{n,m}$ increases. Moreover, the maximum task completion time of Algorithm 2 is significantly lower than the baseline schemes, which substantiates its superior performance. Analyzing Figure 7c, it is apparent that since Algorithm 2 achieves energy reduction with increasing transmission rate, while maintaining the lowest delay among all five algorithms, its weighted ETC is minimized.

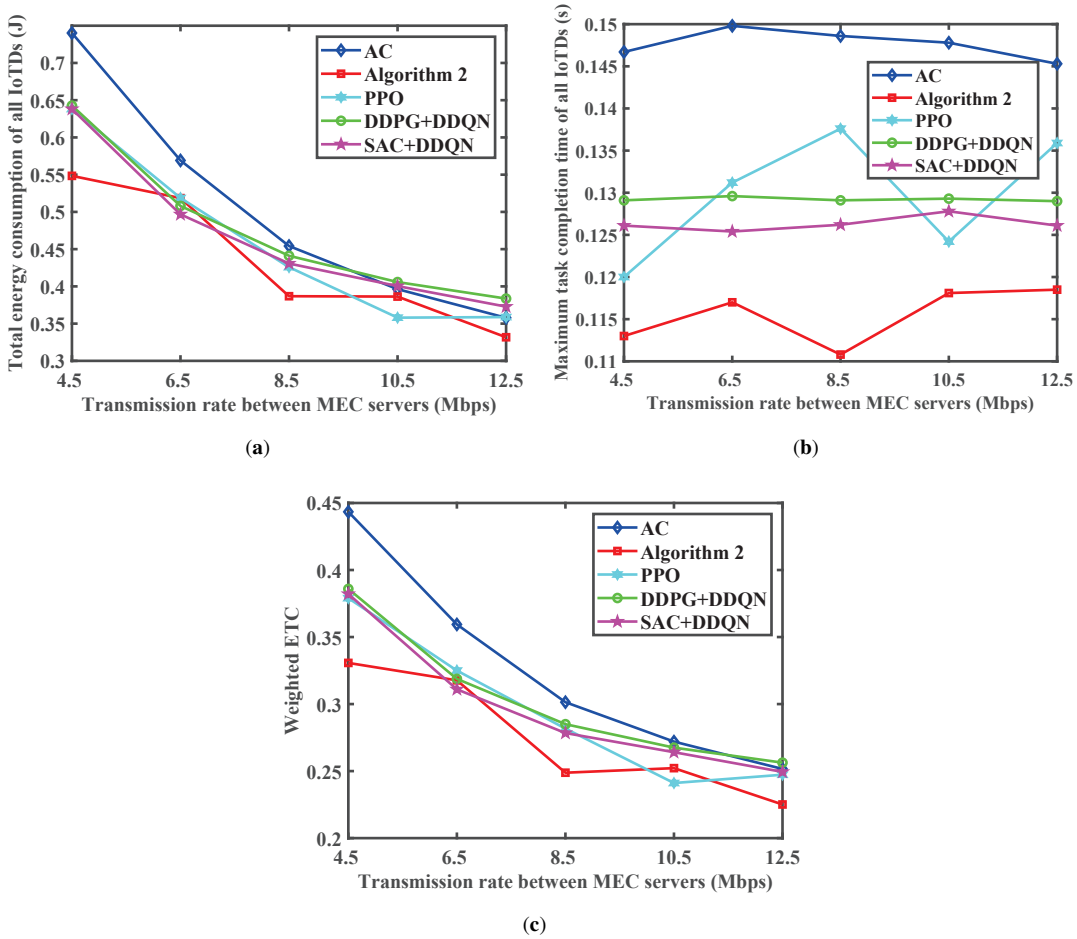

Figure 7. Comparison of the performance between Algorithm 2 and the baseline schemes under different transmission rates between MEC servers. (**a**) Total energy consumption of all IoTDs; (**b**) Maximum task completion time of all IoTDs (s); (**c**) Weighted ETC.

Figure 8 depicts the performance comparison of Algorithm 2 with the four baseline schemes under different transmission rates $R_{n,c}$ between MEC server and cloud. Analyzing Figure 8a, when the transmission rate $R_{n,c}$ increases incrementally by 2 Mbps (from $R_{n,c} = 4.5$ Mbps to $R_{n,c} = 12.5$ Mbps), the total energy consumption of IoTDs gradually decreases as a whole. We can observe the PPO algorithm exhibits lower energy consumption than Algorithm 2 at $R_{n,c} = 4.5$ Mbps, attributable to its characteristic in prioritizing immediate rewards over long-term returns. However, as the transmission rate $R_{n,c}$ increases, Algorithm 2 achieves energy reduction, attaining the minimal energy usage among all the five schemes. This validates Algorithm 2's superior adaptability to dynamic network conditions and its optimal trade-off between immediate and future rewards. Analyzing Figure 8b, it is apparent that the maximum task completion time does not decrease as the transmission rate $R_{n,c}$ between MEC server and cloud increases. Moreover, the maximum task completion time of Algorithm 2 is the minimum. Due to the results in Figure 8c, we can see that Algorithm 2 attains the minimal ETC except for the case when $R_{n,c} = 4.5$ Mbps, where it is marginally outperformed by PPO.

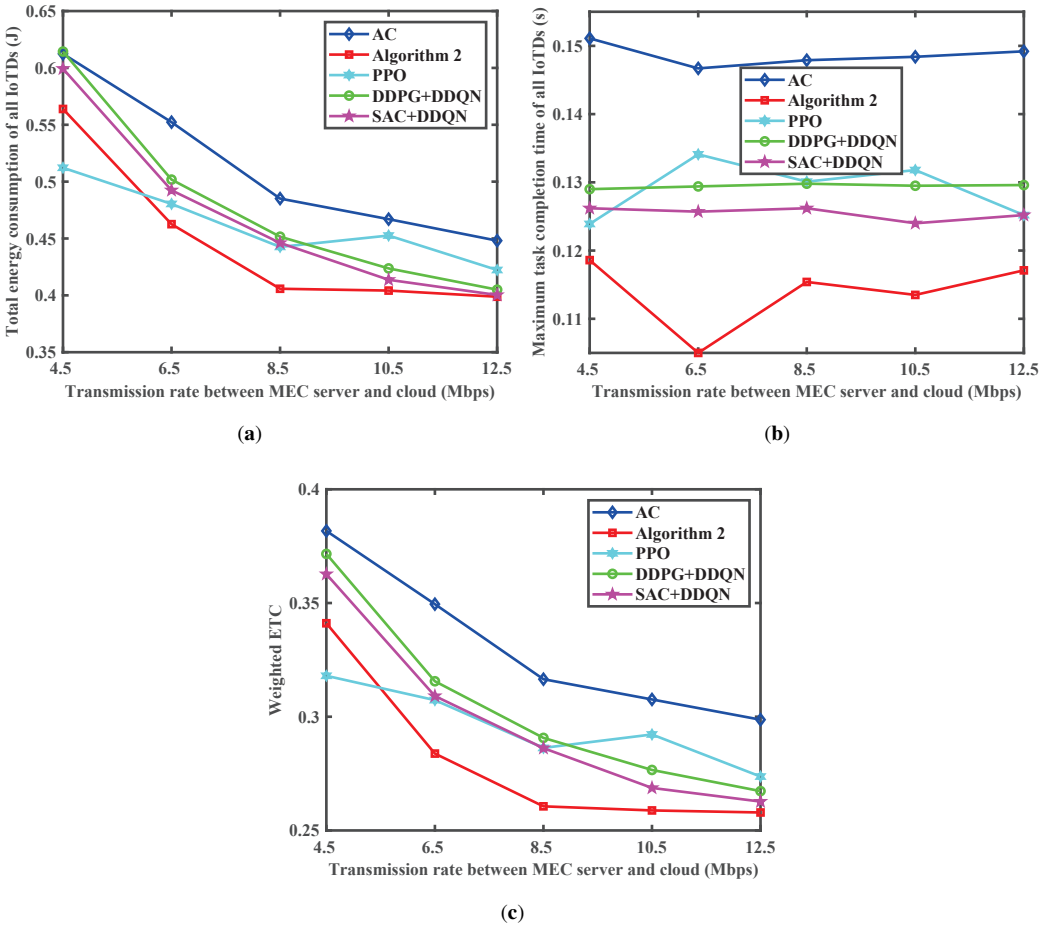

Figure 8. Comparison of the performance between Algorithm 2 and the baseline schemes under different transmission rates between MEC servers and the cloud. (**a**) Total energy consumption of all IoTDs; (**b**) Maximum task completion time of all IoTDs (s); (**c**) Weighted ETC.

Figure 9 demonstrates the performance comparison of Algorithm 2 with the four baseline schemes in connection with delay, energy, and ETC when the number of IoTDs, i.e., K, grows from 4 to 12. Each IoTD has 10 subtasks, and the other parameters maintain consistency with the experimental configuration described in Figure 5. In Figure 9a, we calculate the total energy consumption required for completing all subtasks of the IoTDs. As expected, as K increases, the energy consumption also increases. Moreover, the energy consumption of Algorithm 2 is lower than the four baseline schemes except that when $K = 8$. This demonstrates that A2C achieves the best overall performance. Analysis of Figure 9b reveals that when K rises, so does the delay incurred by subtasks. In all cases, Algorithm 2 demonstrates lower delay compared with the other four baseline schemes. Consequently, as illustrated in Figure 9c, Algorithm 2 exhibits consistently smaller weighted ETC than the four baseline schemes, thereby validating the superior effectiveness of Algorithm 2.

Figure 10 illustrates the performance comparison of Algorithm 2 with the four baseline schemes in terms of delay, energy, and ETC when the number of subtasks I of an IoTD changes from 4 to 12. There exists 12 IoTDs in this experiment, and the other parameters were the same as those for Figure 5. As illustrated in Figure 10a, as I grows, the cumulative energy expenditure of all IoTDs increases. When $I = 4$, Algorithm 2 exhibits higher energy consumption relative to the other four baseline schemes. But, in the other cases, Algorithm 2 consistently achieves lower energy consumption than the baseline schemes. Overall, Algorithm 2 still demonstrates superior performance by comparison with the other baseline schemes. As illustrated in Figure 10b, the delay of all the five algorithms increases with I increases, and Algorithm 2 demonstrates the lowest delay in most cases, which substantiates its superior performance. Consequently, as illustrated in Figure 10c, Algorithm 2 generally achieves the lowest weighted ETC. These results demonstrate that Algorithm 2 outperforms all other baseline schemes with respect to overall performance.

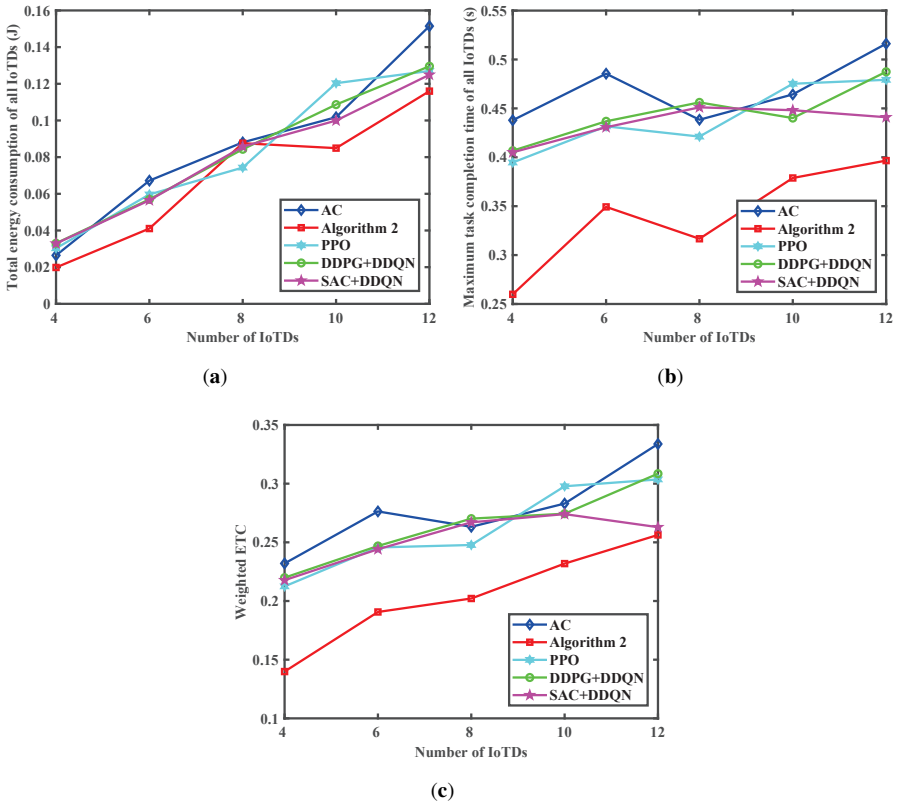

Figure 9. Comparison of the performance between Algorithm 2 and the baseline schemes under different numbers of IoTDs. (**a**) Total energy consumption of all IoTDs; (**b**) Maximum task completion time of all IoTDs (s); (**c**) Weighted ETC.

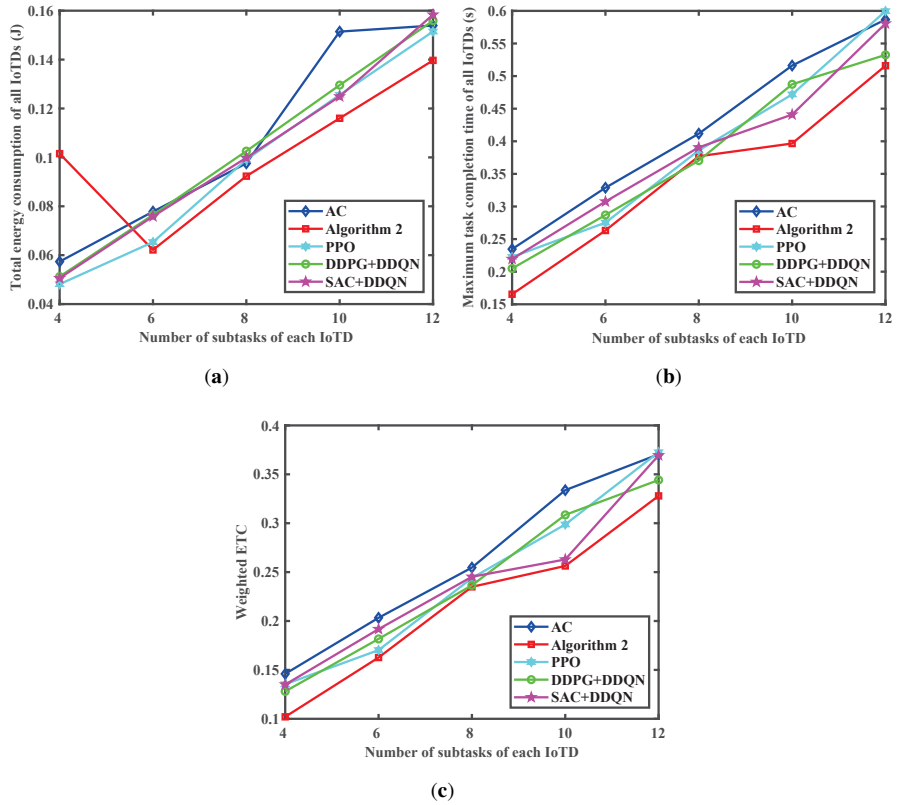

Figure 10. Comparison of the performance between Algorithm 2 and the baseline schemes under different numbers of subtasks of each IoTD. (**a**) Total energy consumption of all IoTDs; (**b**) Maximum task completion time of all IoTDs (s); (**c**) Weighted ETC.

Figure 11 depicts the comparative results of delay, energy consumption, and ETC between Algorithm 2 and the baseline schemes under different caching capacity C_n of MEC server n. Analyzing Figure 11a,b, we can see that as C_n increases from 20 MB to 60 MB, the delay and energy consumption of all the five schemes as a whole gradually decreases and Algorithm 2 has the best performance. Because as the caching capacity keeps increasing, more subtasks can be handled at MEC servers to avoid transferring to the cloud. Figure 11c further illustrates that both time and energy gradually decrease as C_n increases; and then the weighted ETC also gradually decreases. Hence, it is necessary to consider service-caching optimization to improve system performance.

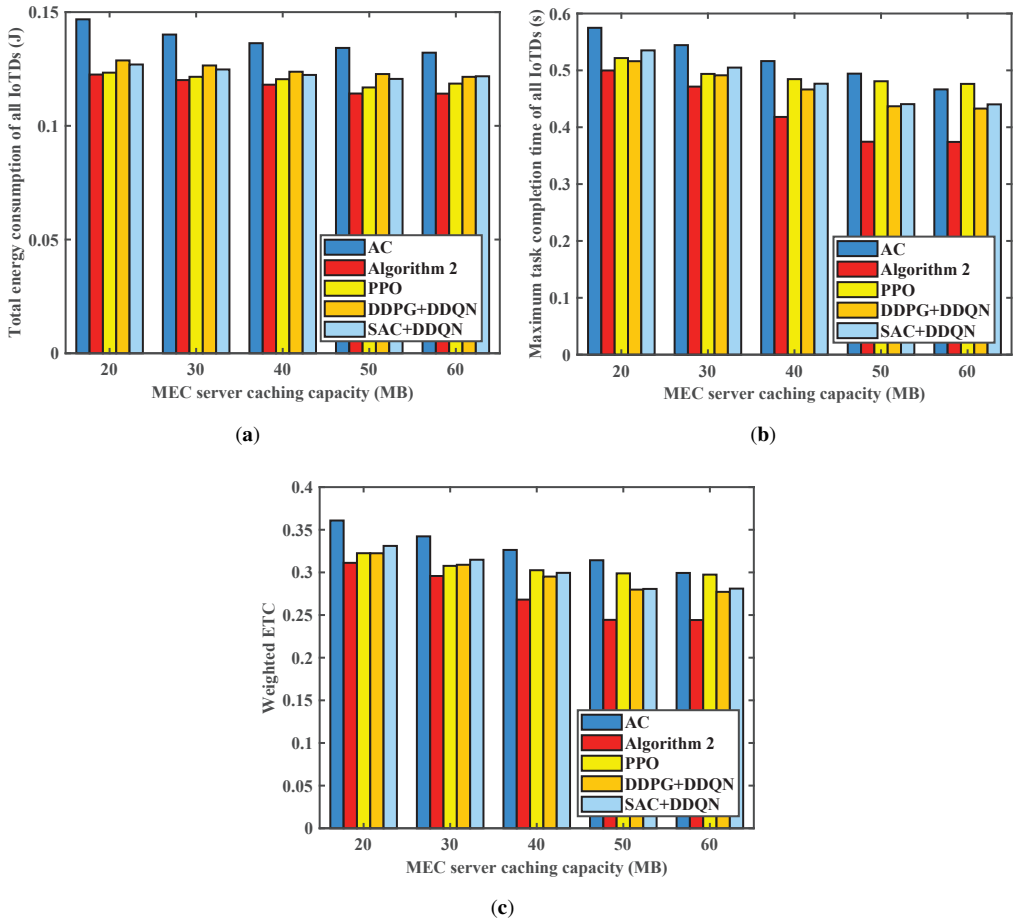

Figure 11. Comparison of the performance of between Algorithm 2 and the baseline schemes under different caching capacity of each MEC server. (**a**) Total energy consumption of all IoTDs; (**b**) Maximum task completion time of all IoTDs (s); (**c**) Weighted ETC.

5. Discussion

To contextualize our contributions, we compare our proposed GCN-A2C scheme with recent publications addressing similar challenges in edge computing.

Dependency-aware offloading with GCN-DRL integration: Several recent works combine graph neural networks with DRL for dependent task offloading. Tong et al. [36] proposed a GCN-PPO framework that extracts DAG features for subtask offloading decisions but does not consider service-caching constraints. Liu et al. [37] introduced MGSAC, combining GCN with SAC for task migration in cloud-edge environments. However, their focus is on task migration rather than joint optimization with caching constraints. In contrast, our work simultaneously manages subtask dependencies and cooperative service-caching in a unified framework. GCN for Network Representation in DRL: NetMind+ [38] demonstrated GCN-based encoding for aggregating dynamic network features in DRL. While their application domain differs, their success validates our methodological choice of GCN for capturing structural dependencies, which we extend to dependency-aware task offloading with caching constraints.

Distinct advantages of our approach: Compared with these state-of-the-art works, our GCN-A2C scheme offers three distinct advantages: (1) Joint optimization of computation-offloading, cooperative service-caching, and resource-allocation under realistic constraints, addressing a more comprehensive problem than works focusing

on individual aspects; (2) Hybrid action space handling that efficiently manages both discrete caching/offloading decisions and continuous resource-allocation within a unified A2C framework; (3) Superior empirical performance with lower ETC, faster convergence, and greater robustness across diverse experimental conditions.

6. Conclusions

In this paper, we developed the joint optimization scheme of dependency-aware computation-offloading, resource-allocation, and collaborative service-caching for MEC systems with multiple MEC servers which cooperatively serve multiple IoTDs with the aid of a cloud. First, considering the constraints of limited caching capacity and computation resources, we formulated a MINLP problem to minimize the weighted ETC of all IoTDs. Then, we modeled the subtasks of each IoTD as a DAG and used GCN to capture the features and dependencies of the subtasks, which are used to assist DRL to obtain the subtask offloading, service-caching, and resource-allocation decisions. Extensive simulations verified the effectiveness of our proposed scheme, demonstrating superior performance compared to baselines. Furthermore, the discussion section compared our work with literature, highlighting the distinct advantages of our joint optimization framework, hybrid action space handling, and superior experimental performance. Finally, in future research, we will consider data processing across multiple time slots and scenarios where subtasks of an IoTD belong to different caching types.

Author Contributions

Y.G.: Conceptualization, Methodology, Software, Validation, Formal analysis, Writing—original draft. F.W.: Conceptualization, Methodology, Supervision, Funding acquisition, Writing—review & editing. All authors have read and agreed to the published version of the manuscript.

Funding

This work was supported in part by Zhejiang Lab under [grant number 2021LC0AB01], in part by the Natural Science Key Foundation of Chongqing under [grant number CSTB2023NSCQ-MSX1017], and in part by the Science and Technology Research Program of Chongqing Municipal Education Commission under [grant No. KJQN202303108].

Institutional Review Board Statement

Not applicable.

Informed Consent Statement

Not applicable.

Data Availability Statement

Not applicable.

Conflicts of Interest

The authors declare no conflict of interest.

Use of AI and AI-Assisted Technologies

No AI tools were utilized for this paper.

References

1. Mao, Y.; You, C.; Zhang, J.; et al. A Survey on Mobile Edge Computing: The Communication Perspective. *IEEE Commun. Surv. Tutor.* **2017**, *19*, 2322–2358.
2. Guim, F.; Metsch, T.; Moustafa, H.; et al. Autonomous Lifecycle Management for Resource-Efficient Workload Orchestration for Green Edge Computing. *IEEE Trans. Green Commun. Netw.* **2022**, *6*, 571–582.
3. Luo, R.; Jin, H.; He, Q.; et al. Cost-Effective Edge Server Network Design in Mobile Edge Computing Environment. *IEEE Trans. Sustain. Comput.* **2022**, *7*, 839–850.
4. Qiu, H.; Zhu, K.; Luong, N.C.; et al. Applications of Auction and Mechanism Design in Edge Computing: A Survey. *IEEE Trans. Cogn. Commun. Netw.* **2022**, *8*, 1034–1058.
5. Li, X.; Xu, Z.; Fang, F.; et al. Task Offloading for Deep Learning Empowered Automatic Speech Analysis in Mobile Edge-Cloud Computing Networks. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1985–1998.

6. Koganti, Y.; Sridhar, V.; Yadav, R.N.; et al. QoS-Aware Application Assignment and Resource Utilization Maximization Using AHP in Edge Computing. *IEEE Internet Things J.* **2025**, *12*, 17717–17728.
7. Zhou, X.; Ge, S.; Liu, P.; et al. DAG-Based Dependent Tasks Offloading in MEC-Enabled IoT with Soft Cooperation. *IEEE Trans. Mob. Comput.* **2024**, *23*, 6908–6920.
8. Deng, X.; Yang, H.; Zhang, J.; et al. Task Offloading in Internet of Vehicles: A DRL-Based Approach with Representation Learning for DAG Scheduling. *IEEE Trans. Mob. Comput.* **2025**, *24*, 5045–5060.
9. Liu, H.; Huang, W.; Kim, D.I.; et al. Towards Efficient Task Offloading with Dependency Guarantees in Vehicular Edge Networks Through Distributed Deep Reinforcement Learning. *IEEE Trans. Veh. Technol.* **2024**, *73*, 13665–13681.
10. Mo, C.-T.; Chen, J.-H.; Liao, W. Graph Convolutional Network Augmented Deep Reinforcement Learning for Dependent Task Offloading in Mobile Edge Computing. In Proceedings of the 2023 IEEE Wireless Communications and Networking Conference (WCNC), Glasgow, UK, 26–29 March 2023; pp. 1–6.
11. Gao, M.; Shen, R.; Li, J.; et al. Computation Offloading with Instantaneous Load Billing for Mobile Edge Computing. *IEEE Trans. Serv. Comput.* **2022**, *15*, 1473–1485.
12. Yan, J.; Bi, S.; Zhang, Y.J.A. Offloading and Resource Allocation with General Task Graph in Mobile Edge Computing: A Deep Reinforcement Learning Approach. *IEEE Trans. Wireless Commun.* **2020**, *19*, 5404–5419.
13. Baghban, H.; Rezapour, A.; Hsu, C.-H.; et al. Edge-AI: IoT Request Service Provisioning in Federated Edge Computing Using Actor-Critic Reinforcement Learning. *IEEE Trans. Eng. Manag.* **2024**, *71*, 12519–12528.
14. Goudarzi, S.; Soleymani, S.A.; Anisi, M.H.; et al. Optimizing UAV-Assisted Vehicular Edge Computing with Age of Information: An SAC-Based Solution. *IEEE Internet Things J.* **2025**, *12*, 4555–4569.
15. Zhou, J.; Zhao, Y.; Zhao, L.; et al. Adaptive Task Offloading with Spatiotemporal Load Awareness in Satellite Edge Computing. *IEEE Trans. Netw. Serv. Manag.* **2024**, *11*, 5311–5322.
16. Zhang, J.; Zhang, G.; Bao, X.; et al. Dependent Application Offloading in Edge Computing. *IEEE Trans. Cloud Comput.* **2023**, *11*, 3439–3451.
17. Liu, J.; Ren, J.; Zhang, Y.; et al. Efficient Dependent Task Offloading for Multiple Applications in MEC-Cloud System. *IEEE Trans. Mob. Comput.* **2023**, *22*, 2147–2162.
18. Chen, J.; Yang, Y.; Wang, C.; et al. Multitask Offloading Strategy Optimization Based on Directed Acyclic Graphs for Edge Computing. *IEEE Internet Things J.* **2022**, *12*, 9367–9378.
19. Liao, Z.; Cheng, S.; Zhang, J.; et al. GpDB: A Graph-partition Based Storage Strategy for DAG-Blockchain in Edge-cloud IIoT. *IEEE Trans. Ind. Inf.* **2025**, *21*, 5790–5798.
20. Wang, Z.; Goudarzi, M.; Buyya, R. TF-DDRL: A Transformer-Enhanced Distributed DRL Technique for Scheduling IoT Applications in Edge and Cloud Computing Environments. *IEEE Trans. Serv. Comput.* **2025**, *18*, 1039–1053.
21. Goudarzi, M.; Palaniswami, M.; Buyya, R. A Distributed Deep Reinforcement Learning Technique for Application Placement in Edge and Fog Computing Environments. *IEEE Trans. Mob. Comput.* **2023**, *22*, 2491–2505.
22. Goudarzi, M.; Rodriguez, M.A.; Sarvi, M.; et al. μ-DDRL: A QoS-Aware Distributed Deep Reinforcement Learning Technique for Service Offloading in Fog Computing Environments. *IEEE Trans. Serv. Comput.* **2024**, *17*, 47–59.
23. Chen, Z.; Wang, F.; Zhang, X. Joint Optimization for Cooperative Service-Caching, Computation-Offloading, and Resource-Allocations over EH/MEC-based Ultra-Dense Mobile Networks. In Proceedings of the 2023 IEEE International Conference on Communications (ICC), Rome, Italy, 28 May–01 June 2023; pp. 716–722.
24. Yan, H.; Xu, X.; Bilal, M.; et al. Customer Centric Service Caching for Intelligent Cyber-Physical Transportation Systems with Cloud-Edge Computing Leveraging Digital Twins. *IEEE Trans. Consum. Electron.* **2024**, *70*, 1787–1797.
25. Chen, Z.; Liang, J.; Yu, Z.; et al. Resilient Collaborative Caching for Multi-Edge Systems with Robust Federated Deep Learning. *IEEE Trans. Netw.* **2025**, *33*, 654–669.
26. Zeng, J.; Zhou, X.; Li, K. Resource-Efficient Joint Service Caching and Workload Scheduling in Ultra-Dense MEC Networks: An Online Approach. *IEEE Trans. Netw. Serv. Manag.* **2025**, *22*, 1788–1800.
27. Zhou, H.; Zhang, Z.; Li, D.; et al. Joint Optimization of Computing Offloading and Service Caching in Edge Computing-Based Smart Grid. *IEEE Trans. Cloud Comput.* **2023**, *11*, 1122–1132.
28. Liu, L.; Chen, Z. Joint Optimization of Multiuser Computation Offloading and Wireless-Caching Resource Allocation with Linearly Related Requests in Vehicular Edge Computing System. *IEEE Internet Things J.* **2024**, *11*, 1534–1547.
29. Li, Y.; Zhu, X.; Li, N.; et al. Collaborative Content Caching and Task Offloading in Multi-Access Edge Computing. *IEEE Trans. Veh. Technol.* **2023**, *72*, 5367–5372.
30. Zhao, Y.; Liu, C.; Hu, X.; et al. Quek. Joint Content Caching, Service Placement, and Task Offloading in UAV-Enabled Mobile Edge Computing Networks. *IEEE J. Sel. Areas Commun.* **2025**, *43*, 51–63.
31. Wei, X.; Cai, L.; Wei, N.; et al. Joint UAV Trajectory Planning, DAG Task Scheduling, and Service Function Deployment Based on DRL in UAV-Empowered Edge Computing. *IEEE Internet Things J.* **2023**, *10*, 12826–12838.
32. Zhao, L.; Zhao, Z.; Hawbani, A.; et al. Dynamic Caching Dependency-Aware Task Offloading in Mobile Edge Computing. *IEEE Trans. Comput.* **2025**, *74*, 1510–1523.

33. Sun, Y.; Zhang, X. A2C Learning for Tasks Segmentation with Cooperative Computing in Edge Computing Networks. In Proceedings of the 2022 IEEE Global Communications Conference (GLOBECOM), Rio de Janeiro, Brazil, 4–8 December 2022; pp. 2236–2241.
34. Lu, J.; Yang, J.; Li, S.; et al. A2C-DRL: Dynamic Scheduling for Stochastic Edge-Cloud Environments Using A2C and Deep Reinforcement Learning. *IEEE Internet Things J.* **2024**, *11*, 16915–16927.
35. Wang, J.; Wang, F.; Zhang, X.; et al. Joint Optimizations for Double-IRS' Cooperative Positioning and Beamforming Over Massive-MIMOAP Based 6G Secure Mobile Wireless Networks. In Proceedings of the 2024 IEEE Global Communications Conference (GLOBECOM), Cape Town, South Africa, 8–12 December 2024; pp. 1–6.
36. Tong, R.; Guo, S.; Qiu, X.; et al. GCN and DRL Based on Dependent Task Offloading Mechanism in Edge Computing. *Digit. Commun. Netw.* **2025**, *12*, 397–404.
37. Wang, Y.; Chen, J.; Wu, Z.; et al. Efficient Task Migration and Resource Allocation in Cloud–Edge Collaboration: A DRL Approach with Learnable Masking. *Alex. Eng. J.* **2025**, *111*, 107–122.
38. Li, H.; Li, P.; Assis, K.R.D.; et al. NetMind+: Adaptive Baseband Function Placement with GCN Encoding and Incremental Maze-Solving DRL for Dynamic and Heterogeneous RANs. *IEEE Trans. Netw. Serv. Manag.* **2025**, *22*, 3419–3432.