A Deep Reinforcement Learning Approach for Dependent Task Offloading in Multi-Access Edge Computing

Bo Xie, Haixia Cui, *Senior Member, IEEE*, Yejun He, *Senior Member, IEEE*, and Mohsen Guizani, *Fellow, IEEE*

***Abstract*—The resource-constrained edge devices are struggling with the rising computational demands of modern user applications with many dependent tasks. In this paper, we investigate the dependent task offloading strategy by constructing graph theory for the dependent tasks so that the explicit priority relations based on deep reinforcement learning (DRL) are drawn. In particular, we propose a novel task-priority-transformer dependent task offloading (TPTDTO) method with the refined priority features of dependent tasks and the application context information. We first construct a priority representation for the dependent tasks which enables the efficient task offloading and energy consumption reduction. To improve the intelligent offloading decision efficiency, a proximal policy optimization (PPO) method is utilized to interact with the multi-access edge computing (MEC) system. Specifically, we leverage the enhanced representation capability of the transformer model to encode the dynamic MEC application topology and utilize a deep neural network to capture the task priority features. Moreover, the transformer model is adopted to address the scalability issue. Simulation results demonstrate that the proposed algorithm can achieve competitive performance compared with that of the existing baselines for dependent task offloading.**

***Index Terms*—Multi-access edge computing, task-priority-transformer dependent task offloading (TPTDTO), proximal policy optimization (PPO), energy consumption, deep reinforcement learning.**

I. Introduction

DEPENDENT task offloading has garnered extensive attention across diverse domains, such as IoT [1], vehicular networks [2], [4], [5], UAV systems [5], [6], [7], and artificial intelligence [8]. Typically, tasks in wireless communications are modeled as a directed acyclic graph (DAG), with task priorities manually determined to establish a sequential queue. This prioritization, often based on computational requirements or task interdependencies within the DAG, ensures successful task completion. Decision-makers then proceed to make offloading decisions for each task or all tasks, considering the current state of the environment. However, due to the intricate task dependency and dynamic nature of environments, it is challenging to make the optimal task offloading decisions so as to achieve an optimal system performance.

There are mainly two strategies for dependent task offloading: sequential offloading [6], [27], [31], [32], where tasks are offloaded one at a time slot as decisions are made, and batch offloading [4], [7], [9], [25], [29], [30], where decisions for all tasks are made simultaneously, followed by a collective wait for outcomes. Traditional optimization strategies for both sequential and batch offloading involve modeling dependent tasks as a DAG and manually extracting priority information to form a queue for decision-making [4], [6], [7], [27]. However, these methods struggle in high-mobility communication environments with big data processing. The challenge lies in deriving optimal values from vast data volumes, compounded by rapidly changing conditions that render these optimizations unstable. The deep reinforcement learning (DRL) technique shows a great ability to capture the dynamic information by interactions with MEC environments based on the deep neural network (DNN). For the dependent task offloading, the DRL can learn a policy that enables to make the suitable offloading decisions and maximize the expected cumulative reward [25], [29], [30], [31], [32]. Furthermore, due to the robustness of DNN, the DRL still effectively performs its tasks when it confronts with the partial information gaps for MEC systems. Therefore, recent research endeavors have increasingly turned to DRL to address the complexities of dependent task offloading [10], [11], [12], [13], [14], [15]. These studies construct the synthetic dependency among the dependent tasks, such as execution sequence and priority, to define the state space for DRL. However, in practice, the number of divisible subtasks varies by application, leading to sequential queues of differing lengths. Furthermore, many approaches must account for the execution order of tasks as a constraint in the objective function to ensure successful task completion.

Received 5 April 2024; revised 14 September 2024 and 8 November 2024; accepted 16 December 2024. Date of publication 19 December 2024; date of current version 8 August 2025. This work was supported in part by National Key Research and Development Program of China under Grant 2023YFE0107900, in part by the GuangDong Basic and Applied Basic Research Foundation under Grant 2024A1515012052, in part by the National Natural Science Foundation of China (No. 61871433, 61828103, 61201255, and 62071306), and in part by the Research Platform of South China Normal University. The associate editor coordinating the review of this article and approving it for publication was J. Yang. *(Corresponding author: Haixia Cui.)*

Bo Xie and Haixia Cui are with the School of Electronic Science and Engineering, School of Microelectronics, South China Normal University, Foshan 528225, China (e-mail: bo.xie@m.scnu.edu.cn; cuihaixia@m.scnu.edu.cn).

Yejun He is with the College of Electronics and Information Engineering, Shenzhen University, Shenzhen 518060, China (e-mail: heyejun@126.com).

Mohsen Guizani is with the Machine Learning Department, Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE (e-mail: mguizani@ieee.org).

Digital Object Identifier 10.1109/TCCN.2024.3520961

2332-7731 © 2024 IEEE. All rights reserved, including rights for text and data mining, and training of artificial intelligence and similar technologies. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.
Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

We explore edge server scenarios with heterogeneous computing capabilities [6] and SoC technology [33], necessitating batch offloading. However, managing communication among dependent tasks in multi-access edge computing systems presents a significant challenge. To tackle this challenge, we utilize service mesh technology, inspired by [27], to aid in task communication, despite it not accounting for task execution order. This approach effectively coordinates interactions among tasks. Additionally, the extent of sequential queue is tied to the application task count, introducing scalability concerns with fluctuating task numbers.

In this paper, we propose a novel approach named as task-priority-transformer dependent task offloading (TPTDTO) algorithm to resolve the above issues by constructing a priority representation for each application task. We transfer the dependent task offloading problem as a Markov decision process (MDP) to minimize the total system energy consumption and processing delay [12]. Based on the observation of MEC systems, a PPO method is developed to train the discrete system actions in terms of the task offloading schedule [34]. Moreover, representing the applications by DAG, we refine the application topology features in order to capture the contextual interdependencies among the computing tasks. To incorporate the task priorities, we employ the DNN as a priority encoder to capture the priority features using the task dependency. The refined features, such as application topology and task priorities, are then fed into the transformer model [16] for the task offloading schedule. To improve the training efficiency, the state and reward are normalized by the Running Mean & Std approach. Moreover, a masked softmax function is adopted to address the scalability issue when the number of tasks changes. To the best of our knowledge, this is the first work that jointly considers the dependent task offloading by priority and transformer. In brief, our main contributions in this paper are as follows:

1) We propose a novel dependent task offloading approach for MEC systems, called as TPTDTO, which constructs a priority representation for each computing task. It enables the efficient offloading and improves the system performance in terms of the total energy consumption and processing delay.
2) We utilize the transformer model to formulate the dynamic system, the scalability issue, and application topology, apply the Running Mean and Std to normalize the state and reward, and use the DNN to capture the task priority features. These contributions improve the performance and convergence speed of the proposed TPTDTO.
3) We employ PPO method to train the TPTDTO directly based on the MEC system state and then output the optimal offloading decision. We also evaluate the performance advantage of the proposed TPTDTO via extensive simulation experiments compared with several existing baseline algorithms.

The rest of this paper is organized as follows. Related works are introduced in Section II. The system model and problem formulation are introduced in Section III, including the network architecture, communication model, computation model, delay and energy consumption model. In Section IV, we address the formulated optimization objective by our proposed dependent tasks offloading algorithm. Computer simulation results are presented in Section V. Finally, Section VI concludes the paper.

II. Related Work

The existing research on the dependent task offloading issue mainly focuses on the task dependencies. In this section, we will introduce them according to the traditional and intelligent tools in details.

A. Traditional Dependent Task Offloading Methods

Typically, tasks are structured as a DAG to prioritize them through rank scores, influenced by environmental and specific task states, while maintaining a prescribed execution order for successful completion. The execution queue is then determined by these rank scores, upon which offloading decisions are made.

Liu et al. [17] propose the Energy Efficiency Co-Task Computational Offloading (ECTCO) algorithm, leveraging Semidefinite Relaxation (SDR) and probabilistic-based stochastic mapping. This approach uses task execution time as the rank score, considering both the execution order and maximum latency constraints. Sahni et al. [18] also use task execution time as the ranking metric but focus on network flow scheduling to address resource contention during offloading, particularly for tasks sent to multi-hop neighboring devices.

Liu et al. [6] explore a scenario where tasks are distributed across vehicles with heterogeneous computing capabilities, featuring various types of computing units. They represent tasks using a DAG and introduce the RFID method, which incorporates a priority changing mechanism alongside a degree-based weighted earliest finish time mechanism. In [26], an online learning algorithm, OL-DTO, is introduced for dependent task offloading in edge computing, designed to handle offloading with unknown system information. It employs Multi-Armed Bandit (MAB) theory and Lyapunov optimization to navigate the complexities of unknown system dynamics. Roy et al. [19] address resource competition among dependent tasks, enhancing offloading efficiency via a real auction mechanism.

Yuan et al. [9] concentrate on splitting applications into multiple dependent subtasks, with some offloaded to a remote server for parallel processing to minimize application processing time. The splitting ratio is further optimized in [25]. The remote server in [9] encompasses both cloud data centers (CDC) and edge servers, enabling parallel processing of dependent tasks across users, edge servers, and CDCs, spatially. In [4], a template-based method is introduced for vehicular cloud (VC)-assisted task scheduling, where a template maps task components to vehicles. Optimal templates are identified using template searching technology based on the task priority, allowing for parallel task processing by the decision maker. Nguyen et al. in [7] break down the task offloading process in MEC-enabled UAV networks into

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

TABLE I
COMPARISON OF FEATURES AND CONTRIBUTIONS IN DEPENDENT TASK OFFLOADING METHODS

Reference	Methodology	Features	Main Contributions
Liu et al. [17]	ECTCO with SDR and probabilistic-based mapping	Task execution time as rank score, considers maximum latency constraints	Energy-efficient offloading for tasks with strict latency requirements
Sahni et al. [18]	Network flow scheduling	Focus on resource contention during offloading to multi-hop neighbors	Optimizes network scheduling to reduce offloading conflicts
Liu et al. [6]	RFID method with priority-based scheduling	Degree-based weighted earliest finish time	Offloading among heterogeneous vehicles, considers task priority dynamics
Roy et al. [19]	Auction-based resource allocation	Resource competition management for dependent tasks	Increases offloading efficiency with real auction mechanisms
Yuan et al. [9]	Parallel task offloading	Task splitting for cloud and edge parallel processing	Reduces application processing time through parallelization on cloud and edge
Nguyen et al. [7]	Subtask merging strategy for MEC-enabled UAV networks	Combines individual subtasks into unified decision vector	Enables efficient simultaneous offloading in UAV networks
Zhang et al. [10]	OSTTD with policy-gradient-based DRL	Models task relationships with adjacency matrices	Learns an optimal policy for offloading without needing a value function
Song et al. [11]	DQN-based multi-objective optimization	Jointly optimizes time, energy, and cost	Efficiently considers multiple offloading objectives
Liu et al. [12]	Actor-critic-based offloading with task priorities	Task priority based on deadline and energy needs	Improves decision-making by incorporating task priority in offloading
Gong et al. [14]	DDPG-based offloading with task priority optimization	Leverages task priority to optimize DDPG parameters	Enhances decision-making by integrating task priorities for valuable experiences
Cao et al. [31]	GCNN-based feature extraction for DRL	Automatically extracts task dependency features with GCNNs	Improves task offloading by enhancing feature extraction of task dependencies
Mo et al. [32]	Attention-enhanced GCNN framework	Integrates attention mechanisms with GCNNs for better feature selection	Boosts task offloading efficiency by focusing on critical task features
Awada et al. [27]	Cloud-native task encapsulation	Utilizes containers and service mesh technologies for scheduling	Enhances simultaneous offloading capabilities at the network level
Ours	Transformer-based DRL framework	Adaptive handling of variable subtask numbers per application; dynamically learns task dependencies; optimized for heterogeneous	Enables efficient simultaneous offloading in dynamic MEC environments, addressing variability in task numbers

two subproblems, employing a strategy of merging individual subtask decision vectors into a unified decision vector to enable simultaneous offloading of multiple subtasks.

Although offloading dependent subtasks simultaneously for parallel processing has significant potential, ensuring that this idea works effectively at the network level is a

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

challenge. However, [27] utilizes cloud-native technologies to encapsulate dependent tasks within containers for scheduling, leveraging cloud-native service mesh technologies. This approach significantly enhances the potential for simultaneous offloading of dependent tasks at the network level.

B. Intelligent Dependent Task Offloading Methods

The DRL-based decision-making mechanisms for dependent task offloading have been extensively studied in recent years. Zhang et al. proposed the OSTTD method in [10], in which they took the dynamic system status as the inputs and iteratively output a sequence of decisions. The OSTTD method models task relationships as adjacency matrices of dependent tasks, capturing essential information. Specifically, the OSTTD method was a policy-gradient-based approach that directly learned a policy without the need to calculate the value function. Song et al. proposed a DQN-based method in [11] that jointly optimized the multiple objectives, including the completion time, energy consumption, and usage charge. To make an efficient offloading decision, they utilized the proposed DQN-based method and took the execution order of tasks into account as the inputs. Liu et al. [12] represented the task dependencies as a DAG and proposed an actor-critic-based method that took the task priorities into account as the inputs. Specifically, the task priorities referred to the assignment of higher priorities for tasks that have longer time intervals before their deadlines or require higher energy consumption for execution. DDPG agents are used to generate task offloading decisions, optimizing DDPG parameters with task priority information to extract valuable experiences in [13], [14], [29]. Unlike the DDPG algorithm, [28] adopts the A2C algorithm for dependent task offloading. DDPG operates on a deterministic policy for continuous action spaces, whereas A2C uses a stochastic policy, fitting discrete action spaces. These studies demonstrate that no algorithm is universally superior; the choice depends on the specific requirements of different edge computing scenarios. Furthermore, [2] explores scheduling fairness in multi-user environments, and [30] examines the impact of task arrival patterns.

In the DRL-based studies mentioned, characteristics of dependent tasks are manually defined before being fed into the neural network as part of the DRL state space. To enhance feature extraction of task dependencies, [31] employs Graph Convolutional Neural Networks (GCNNs) for automatic feature extraction, while [32] incorporates an attention mechanism into the GCNN framework. These studies highlight the efficiency of GCNNs and attention mechanisms in improving task offloading for dependent tasks.

However, due to the complexity introduced by big data, most of the traditional methods struggle with optimizing offloading decisions and fall short with them. Inspired by the prior studies [6], [7], [9], [14], [27], our work immerses the simultaneous offloading decisions for dependent tasks in dynamic MEC environments with DRL and designs a DRL-based framework. Unlike the previous approaches that rely on sequential task prioritization and overlook scenarios where the number of tasks per application varies among users, our primary contribution lies in using a Transformer model to handle scenarios where the number of subtasks per application varies among users, filling a critical gap in the existing literature.

III. System Model and Problem Formulation

In this section, we present the system model and the dependent task offloading optimization problem formulation. First, we give the details of the dependent task offloading process in MEC. Then, the energy model and the corresponding optimization objective for the optimization problem are described. The main notations are summarized in Table II.

A. System Model

Consider an MEC network consisting of numerous edge servers (ESs), $S = \{s_1, s_2, \ldots, s_m\}$, and a set of edge devices (EDs), $U = \{u_1, u_2, \ldots, u_n\}$, with an application $\mathcal{A}$ and multiple dependent computing tasks $\{t_1, t_2, \ldots, t_n\}$. We present a scenario, as illustrated in Fig. 1, and the application is divided into ten dependent computing tasks, where the node 1 represents the input and the node 6 is the output. It is important to note that the application may encompass multiple input and output nodes. So, we assume that one application has only one input and one output [3]. Let $G = (\mathcal{N}, \mathcal{E})$ denote a DAG which represents an application, where the vertex $t_i \in \mathcal{N}$ and the directed edge $e(t_i, t_j) \in \mathcal{E}$ represent the task t_i and the dependency between t_i and t_j, respectively.

Considering the heterogeneous computing capabilities of MEC hosts and the network's ability to transmit multiple data streams concurrently, we focus on a parallel task offloading scenario. The applications can be represented as DAG, which enables the strategic offloading of computational tasks to the edge servers. For example, as shown in Fig. 1, the DAG represents a dependent task structure designed for a real-time video monitoring and anomaly detection system in a smart city context. The six nodes in the DAG correspond to subtasks, with directed edges indicating dependencies. Subtask 1, capturing real-time video, initiates independently, while subtasks 2 and 3, involving image preprocessing and feature extraction, are executed on edge server 1. The results from these tasks are then transmitted to edge server 2, where subtasks 4 and 5 (event recognition and risk assessment) are performed sequentially. By utilizing parallel processing across edge servers, the system optimizes latency and enhances response efficiency for critical events. The final output, representing the risk assessment results, is sent back to the user, enabling timely decision-making in the monitoring process.

Let $\mathcal{X}_{1:n} = [x_1, x_2, \ldots, x_i, \ldots, x_n]$ denote an offloading plan for the tasks in DAG, where n represents the total task number and x_i represents the offloading decision of t_i. Specially, $x_i \in S \cup u_i$ denotes that t_i is offloaded to the edge servers or local host u_i. Let $t_i = \{d_0, r_i, c_i, (l_i^x, l_i^y)\}$ denote the task t_i, where d_0, r_i, and c_i represent the initial data size, output data size, and the required computation resource in million instructions per second (MIPS), respectively. If the task is executed first, d_0 is greater than zero $(d_0 > 0)$. Otherwise, d_0 is equal to zero, i.e., $d_0 = 0$. Assuming that the coordinate,

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

TABLE II
NOTATIONS AND SYMBOLS

Notation	Explanation
U	Set of EDs
S	Set of ESs
$\mathcal{A}$	Application with dependent tasks
t_i	Task of application $\mathcal{A}$
$G = (\mathcal{N}, \mathcal{E})$	A DAG in which represents an application
$\mathcal{N}$	Tasks
$\mathcal{E}$	The dependency between tasks
d_0	The initial data size
r_i	The output data size
c_i	The required MIPS for excuting task t_i
$(l_{u_i}^x, l_{u_i}^y)$	Location of ES u_i
(s_j^x, s_j^y)	Location of ES s_j
$l_{i,j}^t$	The distance between ED and ES
$\mathcal{R}_{i,j}$	Achievable transmission rate of ED
$B_{i,j}$	The allocated bandwidth
$\gamma_{i,j}$	The signal to noise ratio
$g_{i,j}$	The channel gain between nodes
N_0	The noise power density of ED
$\mathcal{R}_s$	Achievable transmission rate of ES
B_s	The allocated bandwidth
P_s	The uploading transmission power
l_s	The distance between ES
g_s	The channel gain between ES
N_s	The noise power density of ES
G_d	The system topology
$\mathcal{V}$	ES or ED
$\mathcal{F}$	The edge btween ED or ES
$w(i, j)$	Transmission time from vertex i to vertex j
$\mathcal{T}_i(t_{1:m})$	Executing time of tasks $t_{1:m}$ on vertex i
τ_j	The executing time of task t_j
f_i^e	The MIPS of ED u_i
f_j^s	The MIPS of ES s_j
p_t	The transmission power
E_{trans}	The total energy consumption for transmitting data
E_{exe}	The total energy consumption for executing task t_i
p^e	The cpu power of ED and ES
E_{te}	The total system energy consumption
T_s	The transmission time

(l_i^x, l_i^y), indicates the location of ED and the EDs randomly move within the coverage area of one AP indefinitely, we can also set the location of ES as (s_j^x, s_j^y) which remains constant. If the output of task t_i serves as the input for task t_j where $i \neq j$, the output data size of t_i and the input data size of t_j are same as r_i. Note that the values of d_0, r_i, c_i, and (l_i^x, l_i^y) are randomly generated during the task offloading process in order to simulate the real-world applications. This hypothesis introduces some additional challenges to the dependent task offloading strategy due to the inherent stochastic nature of system.

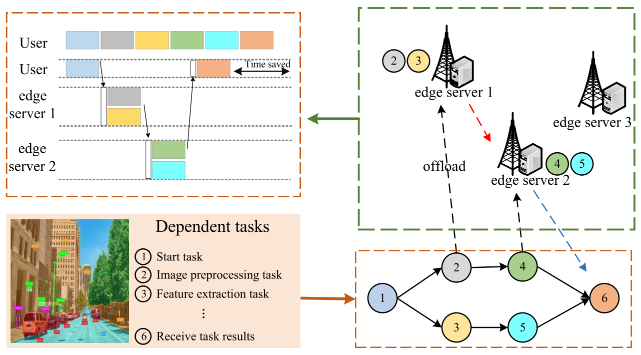


Fig. 1. An example of DAG in wireless communications.

In general, the total system latency consists of transmitting, computing, and receiving time. The sending and receiving times are associated with the network. So, similar to [21], the transmission rate between u_i and s_j can be expressed as

$$\mathcal{R}_{i,j} = B_{i,j} \log_2\left(1 + \gamma_{i,j}\right), \tag{1}$$

where $B_{i,j}$ is the allocated link bandwidth between u_i, and s_j. $\gamma_{i,j}$ is the signal to noise ratio (SNR) which can be given as

$$\gamma_{i,j} = \frac{P_t l_{i,j} g_{i,j}^2}{N_0}, \tag{2}$$

where P_t is the uploading transmission power of u_i to AP. N_0 represents the noise power density. $g_{i,j}$ is the channel gain between u_i and AP. The distance between u_i and s_j can be given by

$$l_{i,j} = \sqrt{\left(l_i^x - s_j^x\right)^2 + \left(l_i^y - s_j^y\right)^2}. \tag{3}$$

The transmission rate between ESs is defined as

$$\mathcal{R}_s = B_s \log_2\left(1 + \frac{P_s l_s g_s^2}{N_s}\right), \tag{4}$$

where l_s is the distance between edge servers by (3). B_s is the allocated link bandwidth between ESs. P_s is the uploading transmission power of ES. N_s and g_s represent the noise power density and channel gain between ESs, respectively.

B. Problem Formulation

We model the transmission time optimization as a shortest path problem with weighted edges in DAG and solve it by the adaptive Dijkstra algorithm, which is shown in Algorithm 1 in detail. Here, let $G_d = (\mathcal{V}, \mathcal{F})$ denote the network topology where the vertex $u_i, s_i \in \mathcal{V}$ and directed edge $(t_i, t_j) \in \mathcal{F}$ represent the ED/ES and data flow from t_i to t_j, respectively. The edge weight, denoted as $w(i, j)$, is the sending time or receiving time if the tasks hold on different vertexes. Note that the proposed TPTDTO is capable of simultaneously

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

determining the offloading decision x_i for all tasks. Thus, the data flow direction between the tasks is known and the transmission time from t_i to t_j can be expressed as

$$w(i,j) = \begin{cases} \frac{r_i}{\mathcal{R}_{i,j}} \\ \frac{r_i}{\mathcal{R}_s} \\ \frac{d_0}{\mathcal{R}_{i,j}} \end{cases}, \tag{5}$$

where $w(i,j) = \frac{r_i}{\mathcal{R}_{i,j}}$ if the data is transmitted between ED and ES, vice versa. $w(i,j) = \frac{r_i}{\mathcal{R}_s}$ if the data is transmitted between different ESs. $w(i,j) = \frac{d_0}{\mathcal{R}_{i,j}}$ if $d_0 > 0$. If the data is transmitted between ED and ES, the corresponding weight is calculated as $w(i,j) = \frac{r_i}{\mathcal{R}i,j}$, where r_i represents the output data size of task t_i, and $\mathcal{R}_{i,j}$ denotes the data the transmission rate between *i* and *j*. Conversely, if the data is transmitted between different ESs, the corresponding weight is given by $w(i,j) = \frac{r_i}{\mathcal{R}s}$, where $\mathcal{R}_s$ represents the data transmission rate between different ESs. Furthermore, $d_0 > 0$ indicates that the task t_i is executed first and the weight is determined as $w(i,j) = \frac{d_0}{\mathcal{R}i,j}$.

Since the computing tasks can be executed parallelly, one vertex can hold more than one tasks. In this case, the executing time of the vertex, $\mathcal{T}_i(t_{1:m})$, is the largest one when the vertex $\mathcal{V}_i$ holds the tasks $t_{1:m}$ and *m* is the task number. Thus, the vertex executing time of the vertex *i*, $\mathcal{V}_i$, can be denoted as

$$\mathcal{T}_i(t_{1:m}) = \max\{\tau_1, \tau_2, \ldots, \tau_j, \ldots, \tau_m\}, \tag{6}$$

where τ_j is the executing time of t_j on vertex $\mathcal{V}_i$. It can be calculated as

$$\tau_j = \begin{cases} \frac{c_i}{f_i^e} \text{ if vertex } i \text{ is ED} \\ \frac{c_i}{f_j^s} \text{ if vertex } i \text{ is ES} \end{cases}, \tag{7}$$

where f_i^e is the computational capacity in terms of million instructions per second or MIPS supported by u_i. Similarly, f_j^s represents the computational capacity provided by s_j. c_i denotes the required computation resource for the execution of task t_j.

We assume that in each time slot, a single user initiates an offloading request for an application comprised of multiple dependent tasks. Communication among these tasks is facilitated by a service mesh, assumed to be pre-installed in the system. This setup allows us to employ a graph shortest path algorithm to estimate the execution time of the dependent tasks.

So far, we can calculate the system transmission time by $T_s = Dijkstra(G_d, n)$ in Algorithm 1, where *n* represents the vertex number in G_d and $\triangleright$ is the comment.

In addition, the system energy consumption is another important parameter which we should consider. It mainly consists of computation and transmission costs. The executing task locally and offloading task to edge servers will result in different energy costs. To briefly, we assume that the system transmission power is constant and thus the transmission cost is determined by the product of the transmission power p_t and time T_s, which can be given by

$$E_{trans} = p_t T_s. \tag{8}$$

Algorithm 1 Adaptive Digkstra Algorithm

```
1: Input $G_d$, the vertex number, $n$;
2: Output the transmission time, $T_s$;
3: $\triangleright$Create an array $\Psi$ with length $n$, initialized to inf.
4: $\Psi = [\psi_1, \psi_2, \cdots, \psi_n], \psi_i = \inf$.
5: $\triangleright$Create an array $\Phi$ with length $n$, initialized to 0.
6: $\Phi = [\phi_1, \phi_2, \cdots, \phi_n]$, $\phi_i = 0$
7: for $i = 0 \to n-1$ do
8:     $x = -1$.
9:     for $j = 0 \to n-1$ do
10:        if $not\ \Phi[j]\ and\ (x = -1\ or\ \Psi[j] < \Psi[x])$ then
11:            $x = j$.
12:        end if
13:        $\triangleright$ Marking the node $x$ as processed
14:        $\Phi[x] = 1$;
15:    end for
16:    for $j = 0 \to n-1$ do
17:        $\Psi[j] = \min(\Psi[j], \Psi[x] + G_d[x][j])$
18:    end for
19: end for
20: $\triangleright$ Find the maximum value in $\Psi$
21: $T_s = \max(\Psi)$
22: $\triangleright$ If the maximum value is maximum integer value
23: if $T_s = \inf$ then
24:     $T_s = -1$.
25: end if
```

Note that T_s encompasses both uplink and downlink transmission times, as detailed in Algorithm 1.

Moreover, the execution energy consumption is determined by the CPU power and the time taken to execute the computing task. However, considering the realistic scenarios in a MEC network, the CPU power consumption $\mathbb{E}[p^e]$ should be modeled as a function of the number of offloading tasks, reflecting the server's resource limitations. Hence, the equation for execution energy consumption can be modified as follows:

$$E_{exe} = \mathbb{E}[p^e(n)] \sum_i^n \tau_i, \tag{9}$$

where $\mathbb{E}[p^e(n)]$ represents the expected CPU power consumption, which is a function of the number of tasks *n*, and $\sum_i^n \tau_i$ is the total execution time.

So, the overall system energy consumption can be formulated as

$$E_{te} = E_{trans} + E_{exe}. \tag{10}$$

Our computing task offloading optimization aims to minimize the long-term energy consumption and system processing latency under some stringent task parameter constraints. Thus, the corresponding optimization problem can be formulated by

$$\min \sum_{j=0}^{n-1} (\lambda E_{te} + (1-\lambda)[\mathcal{T}_i(t_{1:m}) + T_s]) \tag{11}$$

$$\text{s.t. } x_i \in \{0, 1, \ldots, |S|\}, \tag{12}$$

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

$$0 < \sum_{i}^{n} \tau_i \tag{13}$$

$$0 \le \sum_{i,j \in E} w(i,j) \tag{14}$$

where $|S|$ is the edge server number and $\lambda \in [0,1]$ is the scalar weight.

In our decision-making framework, the computing tasks are either processed locally or offloaded to edge servers. Specifically, the constraint (12), where $x_i \in \{0, 1, \ldots, |S|\}$, delineates the possible destinations for task execution. Here, $x_i = 0$ represents local processing within the device itself, while other values correspond to offloading the task to one of the $|S|$ edge servers. Thus, the above objective function is NP-hard [3]. In the subsequent section, we will present a DRL-based dependent task offloading strategy, TPTDTO, for MEC to tackle the aforementioned issues.

Lemma 1: The problem (11) is NP-hard.

Proof of Lemma 1: The proof of this lemma is presented in the Appendix. ∎

IV. Task-Priority-Transformer Dependent Task Offloading Scheduling

In this section, we present the proposed TPTDTO in detail. We first present the necessary background of DRL ingredients including the state space, the action space, and the reward. Then, the overall design process is introduced subsequently.

A. The DRL-Based Task Offloading Model Design

In this paper, we employ a model-free DRL approach, where the system state, action, and reward are explicitly defined. State transitions are not modeled explicitly, as the agent learns the system dynamics through interaction with the environment.

1) System State: System state represents the available information for offloading decisions of agent and also refers to system observation, which can be denoted as $\psi_t \in \mathbb{R}^n$ in time slot *t* by

$$\psi_t = [f_i^e, \mathbf{f}^s, \mathcal{A}_i, \gamma], \tag{15}$$

where f_i^e is the MIPS of u_i and it initiates the offloading request to the control center. $\mathbf{f}^s = [f_1^s, f_2^s, \ldots, f_n^s]$ represents the MIPS set for edge servers where *n* is the number of edge servers. $\mathcal{A}_i = \{t_1, t_2, \ldots, t_n\}$. γ is the corresponding link SNR in Eq. (2).

2) Action: The agent action, $a_t \in \mathbb{R}^n$, can be expressed as

$$a_t = [x_1^t, x_2^t, \ldots, x_k^t], \tag{16}$$

where $x_i^t \in \{0, 1, \ldots, |S|\}$ and $|S|$ is the number of edge servers. $x_i(t) = 0$ represents the local task executing and $x_i^t \in \{1, 2, \ldots, |S|\}$ indicates that t_i is offloaded to edge servers. Here, the subscript *k* represents the application task number.

3) Reward: In the alignment with the optimization objectives of our study, the immediate reward is designed to balance the energy consumption and latency. It can be defined as:

$$r_t = \lambda E_{te} + (1-\lambda)[\mathcal{T}_i(t_{1:m}) + T_s], \tag{17}$$

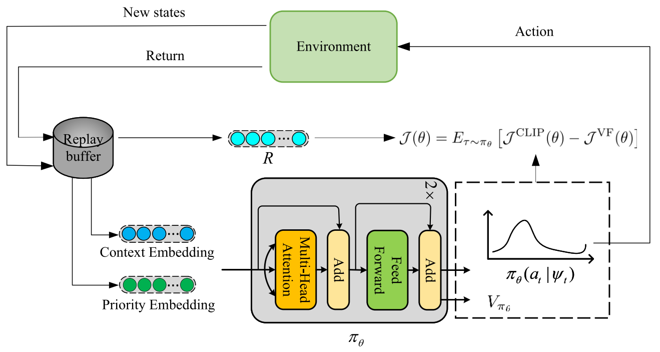


Fig. 2. Our proposed reinforcement learning based task offloading. $\mathcal{J}(\theta)$ is defined in Eq. (34).

where $\lambda \in [0,1]$ is a scalar weight that adjusts the trade-off between energy efficiency (represented by E_{te} and latency, represented by $\mathcal{T}_i(t_{1:m}) + T_s$, ensuring that neither is maximized at the expense of the other. This formulation allows $r_t \in \mathbb{R}$ to reflect the desired compromise between these two crucial aspects of the system.

B. Architecture for TPTDTO

As demonstrated in Fig. 2, the formulated architecture for TPTDTO contains three major components: 1) *State and reward pre-processing*. It utilizes the normalization to improve the convergence speed and PPO model performance; 2) *Priority encoding*. It deploys deep neural network (DNN) as the priority encoder to capture the application priority feature by the edge devices; 3) *Graph representation*. It employs the transformer model as the graph encoder to capture the application topology and context representation.

1) State and Reward Pre-Processing: For the state pre-processing, $\Psi = [\boldsymbol{\psi}_1, \boldsymbol{\psi}_2, \ldots, \boldsymbol{\psi}_i, \ldots, \boldsymbol{\psi}_n]$ is expressed as the state for an episode, where *n* is the episode length and $\boldsymbol{\psi}_i \in \mathbb{R}^m$ is given by Eq. (15). To normalize the episode state, we need to obtain the statistical average value and standard deviation of the state information. However, the mean value and standard deviation may change during the training process. To tackle this issue, we periodically compute the mean and standard deviation of all samples in the replay buffer and then use them as the fixed values in the subsequent updates until the next computation, which is called Running Mean & Std in some existing literature. The mean value and standard deviation can be obtained by

$$\boldsymbol{\mu}^{new} = (1-\tau)\boldsymbol{\mu}^{new} + \tau\boldsymbol{\mu}^{old}, \tag{18}$$

and

$$\boldsymbol{\rho}^{new} = (1-\tau)\boldsymbol{\rho}^{new} + \tau\boldsymbol{\rho}^{old}, \tag{19}$$

respectively, where $\tau \in [0,1]$, $\boldsymbol{\mu} = \mathbb{E}(\Psi)$, $\boldsymbol{\rho} = \sqrt{\mathbb{E}(\Psi - \boldsymbol{\mu})^2}$, $\boldsymbol{\mu} \in \mathbb{R}^{|\boldsymbol{\psi}_i|}$, and $\boldsymbol{\rho} \in \mathbb{R}^{|\boldsymbol{\psi}_i|}$. $|\boldsymbol{\psi}_i|$ is the dimensional of $\boldsymbol{\psi}_i$ and $\mathbb{E}(\cdot)$ is the expectation operation.

So, the normalized state, $\tilde{\boldsymbol{\psi}}_i \in \mathbb{R}^m$, can be defined as

$$\tilde{\boldsymbol{\psi}}_i = \frac{\boldsymbol{\psi}_i - \boldsymbol{\mu}^{new}}{\boldsymbol{\rho}^{new}}. \tag{20}$$

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

The total return from the time slot t to the episode end, $G_t \in \mathbb{R}$, can indicate that the offloading decision is good or not. Since our proposed TPTDTO is based on DNN, the total return will affect the system performance during the training process. In general, the total return can be denoted as

$$\begin{aligned} G_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots \\ &= \sum_t \gamma^{t-1} r_t, \end{aligned} \tag{21}$$

where $\gamma \in [0, 1]$ is the reward discount, $r_t \in \mathbb{R}$ is denoted in Eq. (17).

Considering that if the total return is too large or too small, the neural-based solution during back-propagation will suffer from gradient vanishing or exploding, respectively, we take Eq. (11) as the total return expression and limit it to a certain range [0, 1]. Based on the previous research, we multiply the return by ϵ and rewrite Eq. (21) as

$$\tilde{G}_t = 1 - \epsilon \cdot G_t, \tag{22}$$

where ϵ is a convergence coefficient.

2) Priority Encoding and Graph Representation: Since we assume that the communication between tasks is managed by a service mesh, we need not concern with the execution order characteristics among tasks. Therefore, we directly use the adjacency matrix of the DAG as input to the neural network as context embedding, and the value of each node in the DAG is also used as input to the neural network as priority embedding.

By utilizing the transformer model to capture the priority embedding and according to the system state at time slot t, $\boldsymbol{\psi}_t = [f_i^e, \mathbf{f}^s, \gamma, \mathcal{A}_i]$, we can take the $[\mathcal{A}_i]$ as the graph feature and construct an embedding as the input of the transformer model by

$$\mathbf{z} = [[CLS]; f_i^e; \mathbf{f}^s; \gamma; [SEP]; \mathcal{A}_i], \tag{23}$$

where $\mathbf{z} \in \mathbb{R}^d$ and d is the dimension. [*SEP*] is the token and not utilized in constructing the graph. [*CLS*] is utilized to calculate the state value in the PPO method. $\mathcal{A}_i$ is the application graph topology. The transformer model can be seen as a graph neural network due to its ability to capture the graph-like relationships using self-attention mechanism [22].

By using the adjacency matrix to model the context embedding and given an adjacency matrix, $\mathbf{A} \in \mathbb{R}^{n\times n}$, where n is the number of tasks, we can calculate the in-degree and out-degree by

$$\mathbf{z}_{deg^-}^{-} = \mathbf{A} \cdot \mathbf{I}, \tag{24}$$

and

$$\mathbf{z}_{deg^+}^{+} = \mathbf{A}^T \cdot \mathbf{I}, \tag{25}$$

respectively, where $\mathbf{I} \in \mathbb{R}^n$ is a vector in which all of the elements are 1.

To ensure the stability during the training process, we will scale the in-degree and out-degree by sine and cosine functions when they exceed 2, respectively. Therefore, the input of transformer mode can be denoted as

$$\mathbf{h}^{(0)} = \mathbf{z} + \mathbf{W}_{deg^-} \cdot \sin\left(\mathbf{z}_{deg^-}^{-}\right) + \mathbf{W}_{deg^+} \cdot \cos\left(\mathbf{z}_{deg^+}^{+}\right), \tag{26}$$

where $\mathbf{W}_{deg^-} \in \mathbb{R}^{n\times d}$ and $\mathbf{W}_{deg^+} \in \mathbb{R}^{n\times d}$ are trainable parameters.

So far, we utilize the transformer model to capture the graph and context features by

$$\mathbf{y} = \mathit{Transformer}\left(\mathbf{h}^{(0)}\right). \tag{27}$$

Finally, we compute the probability distribution of the offloading decision for all tasks by

$$\mathbf{y}_{prob} = \mathit{softmax}(\mathbf{y}[1{:}]), \tag{28}$$

where $\mathbf{y}_{prob} \in \mathbb{R}^{n\times m}$, n is the task number, and m is $|S| + 1$. $\mathbf{y}[0]$ is the state value in PPO method.

C. Variable Task Conditions

To address the scalability challenges posed by a variable number of application tasks, we adopt the Transformer architecture, renowned for its proficiency in managing sequences of arbitrary lengths. This adaptability is crucial when the action space is contingent on the fluctuating number of tasks, as it is in our task offloading scenario.

The essence of our solution lies in the utilization of a masking strategy within the softmax function of the Transformer. This strategy effectively neutralizes the influence of irrelevant or non-existent tasks in the action space. We introduce two key operations in our model: a sequence masking function and a masked softmax function. The sequence masking function is defined as follows:

$$M = \begin{cases} M_{i,j} = 1, & \text{if } j < L[i] \\ -\infty, & \text{otherwise} \end{cases}, \tag{29}$$

where M is a negative infinite mask, L is a vector that contains the valid lengths of each sequence in M.

For the masked softmax function, the mathematical representation is given by:

$$\mathit{softmax}(M \odot H), \tag{30}$$

where H is last-layer features of input samples and $\odot$ represents the element-wise multiplication. The $-\infty$ value ensures that the softmax output for masked elements is zero, which prevents these elements from contributing to the output of the softmax operation.

D. Learning Objective for TPTDTO

The policy-gradient (PG) is a tool to control the probability distribution of actions by tuning the policy so as to maximize the return. Given a state, the policy will output an action probability distribution, denoted by a neural network π_θ as

$$\pi_\theta(\psi) = \mathbb{P}[a|\boldsymbol{\psi}; \theta], \tag{31}$$

where π_θ is the transformer model and θ is the trainable parameter.

The PG method can help to optimize the policy π_θ and output a probability distribution over $\pi_\theta(a|\psi)$ which leads to the best cumulative return in (22).

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

To do that, an objective function about the return is defined as

$$\mathcal{J}(\theta) = E_{\tau\sim\pi_\theta}\left[\tilde{G}(\tau)\right]$$
$$= \sum_{\tau} p(\tau;\theta)\tilde{G}(\tau)$$
$$= \sum_{\tau} \tilde{G}(\tau)\prod_{t=0} p\left(\boldsymbol{\psi}_{t+1}|\boldsymbol{\psi}_t, a_t\right)\pi_\theta(a_t|\boldsymbol{\psi}_t). \quad (32)$$

Based on the policy gradient Theorem in [23], we can reformulate the objective function $\mathcal{J}(\theta)$ into a differentiable formalization,

$$\nabla_\theta\mathcal{J}(\theta) = \mathbb{E}_{\pi_\theta}\left[\nabla_\theta \log\pi_\theta(a_t|\boldsymbol{\psi}_t)\tilde{G}(\tau)\right]$$
$$\approx \sum_{t=0}\nabla_\theta \log\pi_\theta(a_t|\boldsymbol{\psi}_t)\tilde{G}(\tau). \quad (33)$$

PPO improves policy performance by limiting divergence from the old policy through objective function clipping [34], ensuring stable enhancements, unlike PG methods where small parameter changes can significantly impact performance[35]. This approach maintains sample efficiency and avoids performance collapse, making PPO a reliable and consistent first-order optimization method.

In this study, the PPO algorithm employs a shared neural network for approximating both policy and value functions. The loss function integrates policy loss and value function loss:

$$\mathcal{J}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\mathcal{J}^{\text{CLIP}}(\theta) - \mathcal{J}^{\text{VF}}(\theta)\right], \quad (34)$$

where $\mathcal{J}^{\text{CLIP}}(\theta)$ is the clipped policy gradient objective, used to limit the pace of policy updates to ensure that the updated policy does not deviate too far from the original policy:

$$\mathcal{J}^{\text{CLIP}}(\theta) = \mathbb{E}_{\tau\sim\pi_\theta}[\min(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}\hat{A}_t,$$
$$\text{clip}(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}, 1-\epsilon, 1+\epsilon)\hat{A}_t)], \quad (35)$$

where $\hat{A}_t = \tilde{G}(\tau) - V_{\pi_\theta}(s_t)$ is the estimate of the advantage function. The clip function, $clip(\cdot)$, and pruning coefficient, ϵ, limit the updating steps and updating amplitude.

$\mathcal{J}^{\text{VF}}(\theta)$ is the value function loss, often measured using mean squared error to assess the difference between the value function prediction and the actual returns:

$$\mathcal{J}^{\text{VF}}(\theta) = \left(\tilde{G}(\tau) - V_{\pi_\theta}\right)^2 \quad (36)$$

To increase the action probability which leads to the energy reduction, we employ the gradient ascent to iteratively update the parameters of neural networks according to

$$\theta \leftarrow \theta + \alpha\nabla_\theta\mathcal{J}(\theta), \quad (37)$$

where α is the learning rate.

Algorithm 2 Task Priority Offloading Scheduling Algorithm

```
1:  **Initialize** the episode replay memory $D$, and $|D| = 100$;
2:  **Initialize** the number of epoch $N$;
3:  **Initialize** the update interval of state mean & std $M$;
4:  **Initialize** the Transformer network with random weight $\theta$;
5:  for $k = 0 \rightarrow N-1$ do
6:      ▷ Collect an episode states.
7:      for $i = 0 \rightarrow |D|-1$ do
8:          $D[i] = (\boldsymbol{\psi}_i, r_i, a_i, \boldsymbol{\psi}_{i+1})$
9:      end for
10:     ▷ Preprocess the states and rewards.
11:     Calculate the mean of the state $\boldsymbol{\mu}$ on $D$.
12:     Calculate the std of the state $\boldsymbol{\rho}$ on $D$.
13:     Normalize the states using the (20).
14:     Calculate the expected return using the (22).
15:     ▷ Encode the priority feature.
16:     $\mathbf{h}^{(0)} = \mathbf{z} + \mathbf{W}_{deg^-}\cdot\sin(\mathbf{z}^-_{deg^-}) + \mathbf{W}_{deg^+}\cdot\cos(\mathbf{z}^+_{deg^+})$
17:     if $k\%M == 0$ then
18:         Update the $\boldsymbol{\mu}$ and $\boldsymbol{\rho}$ using (18) and (19)
19:     end if
20:     ▷ Forward propagation of the policy model;
21:     $Tramsformer(\mathbf{h}^{(0)})$
22:     ▷ Back-propagate the policy model and update the
        parameters of the policy model;
23:     $\theta \leftarrow \theta + \alpha\nabla_\theta\mathcal{J}(\theta)$
24: end for
```

E. Training Process and TPTDTO Design

The detail of the proposed task offloading algorithm is summarized in Algorithm 1, where ▷ is the comments. Firstly, the episode replay memory, the epoch number, the update interval of state mean & std, and the weights of transformer network are initialized. The maximum length of episode replay memory is set to 100, which means that we offload 100 applications in an episode. For each epoch step, we should collect the transition segments $(\boldsymbol{\psi}_i, r_i, a_i, \boldsymbol{\psi}_{i+1})$ for an episode for back-propagation. The action a_i is sampled based on the probability distribution in TPTDTO and it is pre-trained with the input $\mathbf{h}^{(0)}$, which is pre-processed from line 11 to line 13 in Algorithm 1. Note that the mean value and standard deviation of the states are updated at regular intervals. Finally, the updating neural network is improved by conducting the mini-batch Stochastic Gradient Ascent with the target function defined in Eq. (34) for N epochs.

V. Simulation Results

In this section, the simulation results are presented to illustrate the effectiveness of the proposed dependent task offloading algorithm in multi-access edge computing system. The energy-efficient dependent task offloading algorithm and network architecture are implemented with PARL,[1] and the system model is formulated by Python 3.7.

[1]https://github.com/PaddlePaddle/PARL

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

A. Simulation Settings

We implement the proposed TPTDTO on a Linux workstation with 64-bit Ubuntu 22.04.1. The hardware for training all DRL baselines has one Nvidia's GPU with GeForce RTX 3090Ti with 24-GB memory. The CPU is an Intel Core i9-10980XE processor, 18 cores, and 3.00GHz clock speed. Considering a small cell network consisting of three APs and one ES, where the EDs with splittable tasks are randomly moving at each time slot within the coverage of APs, i.e., $200(m) \times 200(m)$. In our experiments, the ED number is 5 and the ES number is set as 3. Moreover, we randomly generate 100 applications with various dependent task numbers like 5, 10, 15, 20, and 25 for an episode.

Similar to [24], the channel gain is modeled as $127 + 30\log(L)$, where L is the transmission distance. One episode has 100 splittable tasks to offload as $C = 100$. The attributes of the task, denoted as t_i, are chosen randomly from a discrete uniform distribution. In order to establish the uniformity in the dynamic environment for each baseline, we have fixed the random seed at 123. The learning rate given by (37) is set as $\alpha = 0.001$ and the reward discount is set as $\gamma = 0.9$. The number of the transformer layers is set as $nh = 2$. We trained all DRL baselines for 25000 epochs, conducting testing every 10 epochs. To investigate the scalability challenges associated with a varying number of application tasks, our experimental setup includes a range of scenarios with the number of application tasks set to vary from 5 to 25.

For the DAG generation, we follow the method described in [15], which allows us to control the properties of the DAG through several parameters, including *fat*, *density*, and *ccr*. The *fat* parameter determines the width and height of the DAG, affecting how many tasks can be executed concurrently and the number of levels in the DAG. The *density* parameter influences the number of edges between different levels of the DAG, thereby impacting the connectivity and dependencies between tasks. The *ccr* parameter, or communication-to-computation ratio, adjusts the balance between communication overhead and computational workload in the DAG. In our simulation experiments, we set *fat*, *density*, and *ccr* to 1, 1, and 0.5, respectively, to reflect typical scenarios in dynamic MEC environments. The DAG generation tool can be accessed from the code repository.[2]

B. Benchmark Algorithms

For comparison, we also evaluate the performance of the following benchmark algorithms except for our proposed *TPTDTO*: *Random*, *RoundRobin*, *WeightGreedy*, *PGDNN* [10], *PGTF*, and *A2C* [28]. Overall, they are described as follows.

1) *Random:* The offloading decision for each task in the DAG is random.
2) *RoundRobin:* The tasks in the DAG make offloading decisions based on the CPU and energy utilization of the local processors and MEC hosts.
3) *WeightGreedy:* Compared to *RoundRobin*, *WeightGreedy* takes into account the weightage of the utilization.
4) *PGDNN* [10]*:* A DNN-based policy gradient method without priority embedding.
5) *PGTF:* A policy gradient method, where we replaced the DNN architecture in PGDNN [10] with Transformer models to evaluate the performance improvements brought by the Transformer-based architecture. This modification is not from the original [10], but was proposed in this paper to validate the effectiveness of Transformers.
6) *A2C* [28]*:* Using the A2C algorithm for dependency task offloading.

[2]https://github.com/TILhub/OCT-Algorithm

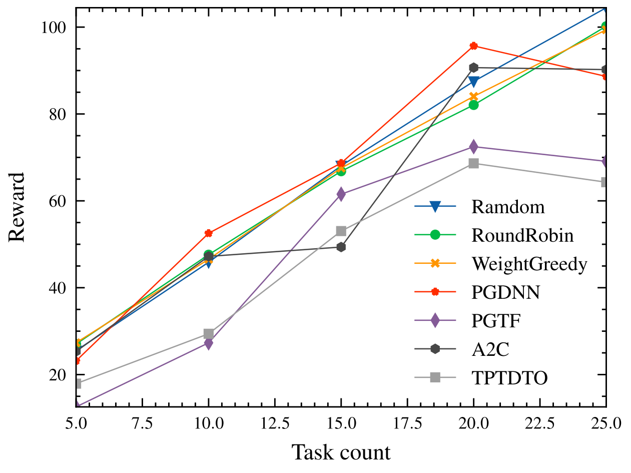


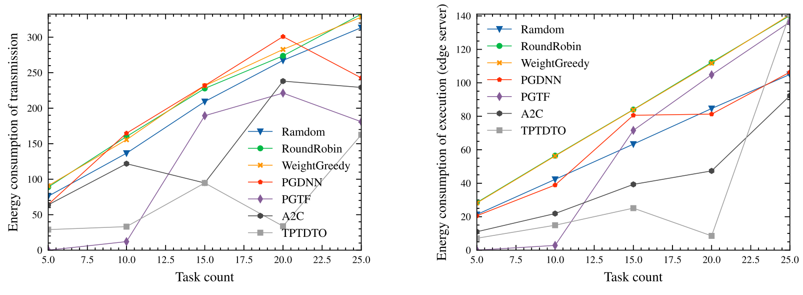
Fig. 3. Average reward for the proposed algorithm and some benchmark algorithms versus task count.

C. Performance Comparisons and Analysis

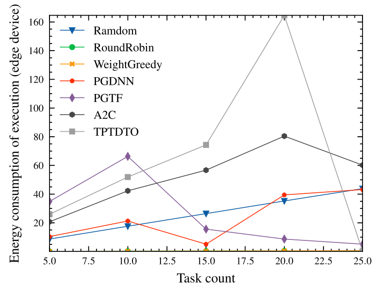
Fig. 3 illustrates the reward performance with different computing task counts of the considered algorithm. From the figure, it is observed that the lower reward indicates the better performance. This is because that the total reward is calculated as the sum of energy consumption and delay, which encompasses both the executing and transmission delays. The experimental results demonstrate that our proposed *TPTDTO* performs better than the other baselines and this verifies the effectiveness of *TPTDTO* in the dependent task offloading. Furthermore, the simulation results of *Random*, *PGDNN*, *RoundRobin*, and *WeightGreedy* are similar significantly. This indicates that the *PGDNN* is not capable of modeling dynamic environments. After replacing the DNN with the transformer model, the policy-gradient method achieves a significant improvement. It demonstrates that the transformer model is better suitable to model the dynamic environments than the traditional DNN. In addition, in comparison with *PGTF*, our *TPTDTO* achieves achieve a better performance result, which indicates that the exploring priority information to model the dependent tasks can improve the computing task offloading efficiency.

Fig. 4(a), 4(b), and 4(c) depict the energy consumption in the network transmission, edge computing, and local computing, respectively. From the figures, we see that the network

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

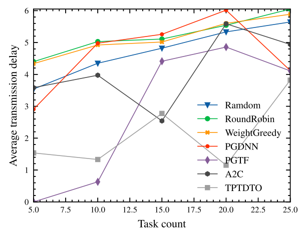


(a) Energy consumption of transmission. (b) Energy consumption of edge servers.

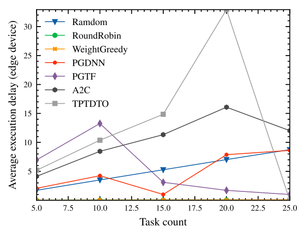


(c) Energy consumption of edge devices.

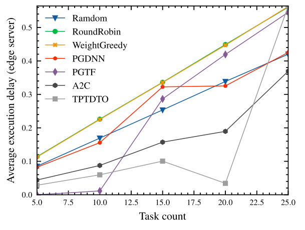
Fig. 4. Energy consumption of transmission, edge servers, and edge devices for the proposed algorithm and some benchmark algorithms.



(a) Average transmission time.



(b) Average execution delay on edge devices.



(c) Average executing time on edge servers.

Fig. 5. Average delay of transmission, edge devices, and edge servers for the proposed algorithm and some benchmark algorithms.

transmission consumes the most energy, followed by the computing tasks at edge servers, which are similar to the other baseline algorithms. This indicates that all baselines tend to offload the computing task to edge servers for execution. In addition, since both *RoundRobin* and *WeightGreedy* use the greedy algorithms which cannot make reasonable offloading decisions for randomly generated tasks, they offload all of their computing tasks to edge servers. Compared with the baselines, our proposed *TPTDTO* achieves a better balance between ESs and EDs by reducing the energy consumption of network transmission. When the task number is set as 15 and 20, respectively, the proposed *TPTDTO* tends to offload the tasks to the local devices for offloading, resulting in higher energy consumption compared to the other baselines for edge devices. When the task count reaches 25, the TPTDTO algorithm shifts its offloading strategy, prioritizing edge servers over local devices. This change reduces the energy consumption and execution delay on edge devices, as seen in Fig. 4(c) and Fig. 5(b), but increases the energy and delay on edge servers and transmission, as shown in Fig. 4(a), 4(b), 5(a), and 5(c).

In order to further analyze the network task transmission and executing time on ESs and EDs, we can refer to the numerical results in Fig. 5(a), Fig. 5(b), and Fig. 5(c). The baseline algorithms tend to offload the computing tasks to edge servers, which may increase the network transmission time and energy consumption. In contrast, the proposed *TPTDTO* offloads the computing tasks to edge devices, leading to a higher executing time. To analyze the combined energy consumption and system delay, it is evident that simultaneously optimizing the two parameters will bring out lots of challenges. However, as being based on neural networks, the proposed *TPTDTO* lacks theoretical guidance to analyze the underlying reasons.

It is important to note that the energy consumption and computation delay are closely related, as demonstrated by the similarity between Fig. 4(b), 4(c) and Fig. 5(b), 5(c). This correlation occurs because the energy consumed by edge devices and servers is directly tied to the computational work they perform, which influences the delay experienced during task execution. As more tasks are offloaded to edge servers,

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

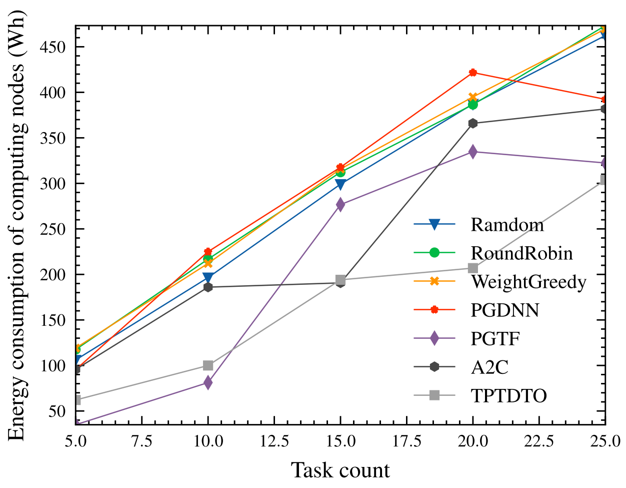


Fig. 6. Energy consumption of computing tasks under different task counts for the proposed algorithm and some benchmark algorithms.

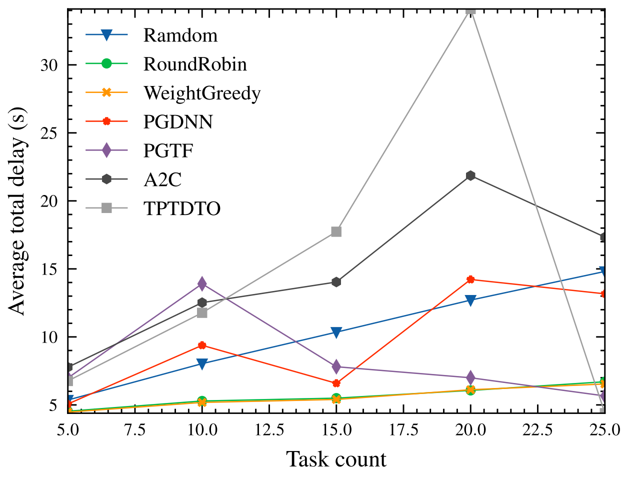


Fig. 7. Average total delay under different task counts for the proposed algorithm and some benchmark algorithms.

both the energy consumption and execution delay on these servers increase simultaneously.

Fig. 6 shows the total energy consumption of computing nodes for task offloading under different task counts. As can be observed, when the task number for an application is fewer than 10, *PGTF* demonstrates better performance compared to our proposed *TPTDTO*. However, when the task number exceeds 10, our proposed algorithm exhibits superior performance over *PGTF*. This result suggests that a smaller number of tasks entail a lesser task topology information. In comparison to *PGTF*, *TPTDTO* utilizes (26) to extract the priority features among tasks. Given 5 and 10 tasks, the energy consumption of *PGTF* is 82% and 79% of the one of *TPTDTO*, respectively. Moreover, when increasing the task number of an application from 15 to 25, the energy consumption of *TPTDTO* is 70%, 64%, and 95% of the one of *PGTF*, respectively.

Fig. 7 illustrates the total system delay performance under different task counts. From this figure, it can be seen that if the task number is 25, our proposed *TPTDTO* can significantly reduce the system's delay compared to the baselines. However, when the task number is less than 25, the performance of the

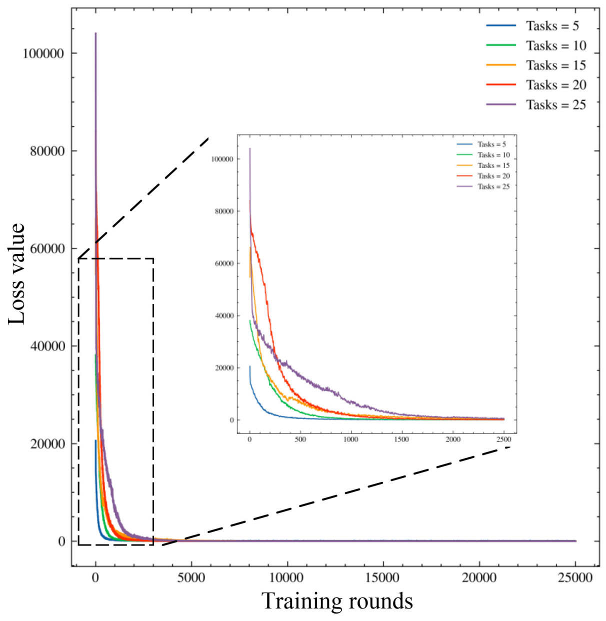


Fig. 8. Convergence diagram of TPTDTO algorithm.

proposed *TPTDTO* is inferior to them. This is because that the proposed *TPTDTO* tends to offload the tasks to local devices due to their limited computational capability. The above simulation results indicate that our proposed *TPTDTO* outperforms the baseline algorithms in reducing system energy consumption. In addition, it is beneficial for the energy efficiency performance by extracting priority information among the application computing tasks and furthermore the transformer model is better for modeling the random systems compared with DNN. Note that with 20 tasks, TPTDTO shows a higher average total delay than other methods due to prioritizing local execution for most tasks, as shown in Fig. 4(a), 4(b), and 4(c).

D. Evaluation of Train Performance

In Fig. 8, we show the relationship between the convergence performance of training loss value for TPTDTO and the varying task number during the intelligent training process. In this case, we observe that as the training time grows, the TPTDTO demonstrates an accelerated convergence, which indicates its outstanding capacity ability to learn the information of system and computing tasks. In particular, Fig. 9 illustrates the return analysis of convergence performance during testing, where the returns are recorded every n steps and n is set to 10. Obviously, the simulation results of the returns exhibit an ascending trend over differing task counts and they substantiate our assertions about the algorithm design effectively. In addition, we also notice that the return value decreases sharply when the task number is set as 25 at the beginning and steadily grows after 100 episodes. When the task number is set as 15, the return value decreases steadily at the beginning and steadily grows after 500 episodes. After 2300 episodes, it decreases sharply. Although we use many useful tricks, there is no theoretical guarantee that the stable return value achieves the optimal policy. This is because that the intelligence of neural networks

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

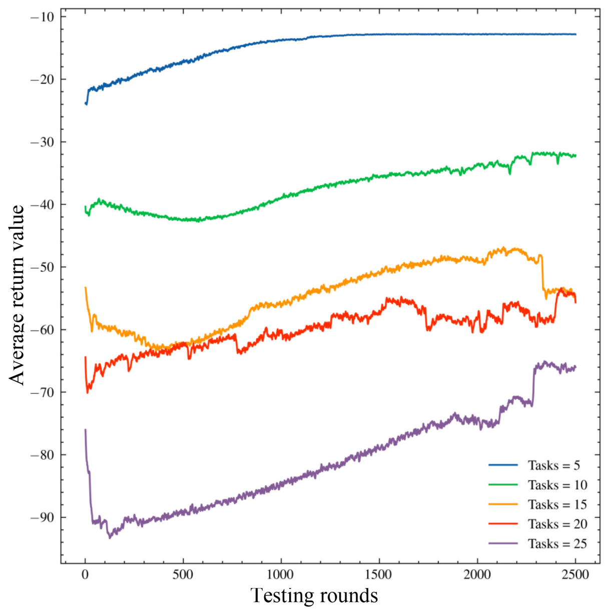


Fig. 9. Convergence performance of returns during testing. Note that DRL always reports return in a higher-is-better format. For this purpose, return is multiplied by −1.

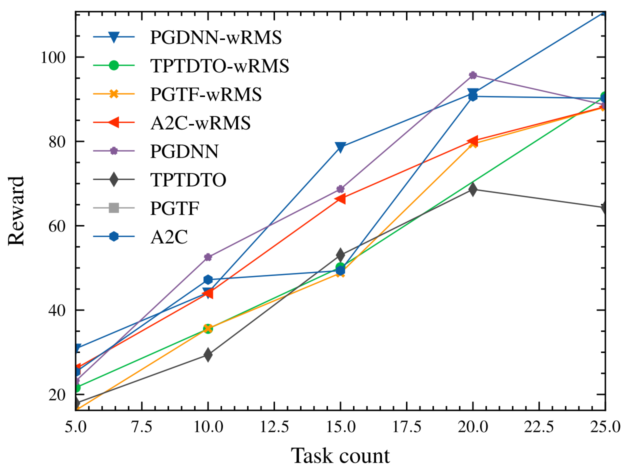


Fig. 10. Comparative experimental results of rewards with and without the use of Running Mean & Std for the proposed algorithm and some benchmark algorithms.

cannot be quantified. Overall, our proposed algorithm can increase the return value.

E. Effect of Running Mean & Std

To evaluate the impact of the Running Mean & Std on our dependent task offloading scenario, we conduct experiments with and without the use of Running Mean & Std, and show the results in Fig. 10. From Fig. 10, it is evident that the strategies employing Running Mean & Std (without the 'wRMS' tag) consistently outperform their counterparts across varying task counts. The TPTDTO curve, which accounts for the Running Mean & Std, demonstrates a lower reward trajectory compared to its non-normalized counterpart, indicating an improved optimization in terms of energy consumption and latency. Further examination reveals that the normalized versions of all strategies, including TPTDTO, PGTF, and PGDNN, exhibit a downward trend in reward values as the task count increases. This trend is consistent with the expected behavior, where the normalization aids in mitigating the variance in reward signals, thereby facilitating a more stable and reliable convergence during the training phase of the DRL model. In contrast, strategies without the Running Mean & Std normalization (denoted with 'wRMS') show an inverse relationship, with the reward increasing alongside the task count, suggesting a less effective optimization process. Particularly for the TPTDTO-wRMS variant, the performance decrement is pronounced, underscoring the significance of the normalization technique in handling higher-dimensional state spaces associated with larger task counts. The disparity in performance between normalized and non-normalized strategies becomes more distinct as the task count exceeds the 15-task threshold. This observation implies that our normalization technique, Running Mean & Std, is particularly beneficial in scenarios with a substantial number of tasks, thereby confirming its role in improving the scalability of the offloading strategies.

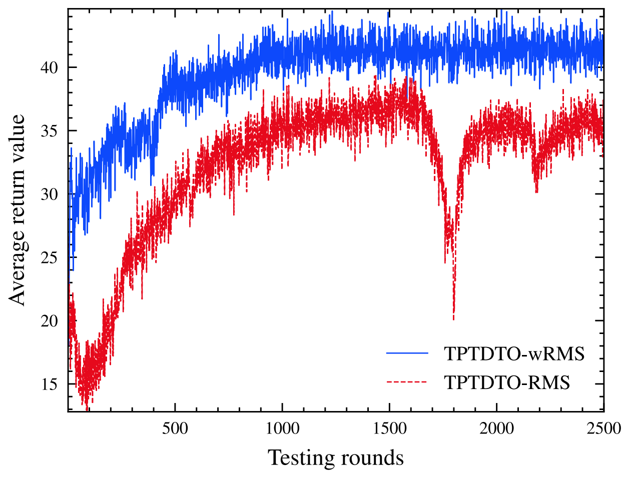


Fig. 11. Experimental results for varying numbers of application tasks as the task count changes.

F. Analysis of Variable Task Conditions

To analyze the scalability of our task offloading strategy as the number of application tasks varies, we conduct experiments on the vary number of tasks range from 5 to 25, and show the results in Fig. 11. Fig. 11 compares the average return values over a range of testing rounds for the TPTDTO strategy with and without the Running Mean & Std normalization technique (denoted as TPTDTO-wRMS and TPTDTO-RMS, respectively). Observing the trends, it is evident that TPTDTO-RMS maintains a lower average return value throughout the testing rounds, which is indicative of a better performance as per our optimization criteria where lower values are preferable. This demonstrates the strategy's robustness and its ability to scale effectively with a changing number of tasks.

Notably, the TPTDTO-wRMS variant exhibits higher variability and overall higher return values, suggesting reduced efficiency and stability when adapting to variable task loads.

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

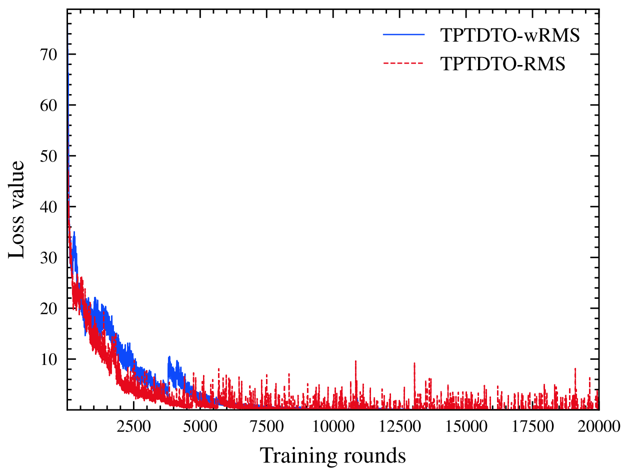


Fig. 12. Comparison of the loss values in the convergence diagram of the TPTDTO algorithm with and without the use of Running Mean & Std.

The inflection point observed in the Fig. 11 for both strategies around the 1500-round mark indicates an environment with a potentially increased number of tasks or a change in task complexity. Here, TPTDTO-RMS shows resilience to these changes, with a smaller increase in return value compared to the TPTDTO-wRMS, highlighting the benefits of the normalization technique in maintaining performance in the face of task variability.

The loss value trend, as shown in Fig. 12, during training rounds indicates that the TPTDTO strategy utilizing the Running Mean & Std normalization (TPTDTO-RMS) converges faster and to a lower loss value than the version without it (TPTDTO-wRMS). This suggests that the normalization technique not only aids in stabilizing the training process but also contributes to the efficiency of the learning algorithm, as evidenced by the reduced loss values. Additionally, the fluctuation in loss for TPTDTO-wRMS indicates less stable convergence, reinforcing the value of including Running Mean & Std in the training process for a more consistent and reliable model optimization.

G. Time Complexity and Space Complexity

To evaluate the time and space complexity of the proposed algorithms and models,[3] we analyze the data presented in Tables III to V. Table III provides a comparison of training times across various baseline models, including both the forward and backward propagation stages. The proposed TPTDTO method is included to assess its computational efficiency relative to other approaches, such as PGDNN, PGTF, and variations of A2C. The training time for TPTDTO, while not the shortest among the methods, is justified by its enhanced capability to handle complex tasks that baseline methods may struggle with.

Table IV presents the parameter sizes of the DNN and Transformer models. As shown, the Transformer model has a slightly larger parameter size than the DNN, indicating a modest increase in memory and storage requirements. This difference reflects the inherent architectural complexities of the Transformer model, which offer additional performance benefits despite its increased parameter size.

[3]The GPU used for evaluation is an NVIDIA 3090Ti. Model parameter sizes were obtained using the ‘summary()‘ function from the PaddlePaddle framework. Both training and inference times were measured with the Paddle model directly, without conversion to TensorRT or model quantization.

TABLE III
COMPARISON OF TRAINING TIME PER ROUND FOR DIFFERENT ALGORITHMS

Algorithm	Train Time (ms)
PGDNN	1.5
PGTF	3.0
A2C	3.8
TPTDTO	6.4

TABLE IV
MODEL SIZE OF DNN AND TF MODELS

Model	Model Size (MB)
DNN	0.37
TF	0.39

TABLE V
INFERENCE TIME FOR DNN AND TF MODELS (MILLISECONDS)

Batch Size	DNN	TF
1	0.252	1.046
100	0.296	1.177
200	0.290	1.150
300	0.294	1.131
400	0.292	1.257
500	0.291	1.262

Table V reports the inference times for the DNN and TF models across varying batch sizes. Given that deep reinforcement learning primarily relies on inference after the policy is learned, this aspect is particularly relevant for assessing deployment efficiency. The results show that the Transformer model exhibits longer inference times compared to the DNN across all tested batch sizes, highlighting its higher computational complexity. However, both models demonstrate scalability, with inference times that remain stable as batch sizes increase.

In conclusion, the proposed TPTDTO method demonstrates effective performance within the application scope, balancing complexity with functionality. Although its time and space complexity may not be optimal, the results indicate that it performs reliably and efficiently, making it well-suited for scenarios that prioritize performance alongside manageable resource demands.

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

VI. Conclusion

In this paper, we have investigated the DRL based dependent task offloading design for multi-access edge computing systems and proposed a PPO approach for dependent task offloading with attention-based model to deal with the energy consumption and delay problem. We have conducted a stochastic MEC system to simulate the actual world and the dependence relationships between the splittable tasks and their properties are randomly generated. This is based on our observation that there are many complex and interdependent tasks in real-world applications. In addition, we have improved the convergence speed and performance of the PPO model by employing normalization techniques while capturing the priority features of dependent tasks. We also use the transformer model as the graph encoder to capture the topology and context representations of the applications. Future work mainly considers to explore the resource competition issues for the application placements, addressing the sensitivity of PPO for further system latency optimization and developing dynamically adaptive neural networks that can adjust the input and output dimensions in response to evolving the infrastructure demands to efficiently manage the fluctuating action spaces.

Appendix
Proof of Lemma 1

To prove that the optimization problem formulated in (11) is NP-hard, we will establish a reduction from a known NP-hard problem, such as the Knapsack Problem, to our task offloading problem. This will demonstrate that solving the task offloading problem is at least as hard as solving the Knapsack Problem, thereby proving that it is NP-hard.

First, consider a simplified version of the problem where the objective is solely to minimize the total energy consumption E_{te}. We disregard the latency term and the weighting factor λ for this initial step. The simplified problem becomes:

$$\min \sum_{j=0}^{n-1} E_{te}, \tag{38}$$

where $E_{te} = E_{trans} + E_{exe}$, representing the energy consumed during transmission and execution.

The Knapsack Problem is a well-known NP-hard problem where the objective is to maximize the total value of items packed into a knapsack without exceeding its weight capacity. Formally, the problem can be stated as:

$$\max \sum_{i=1}^{n} v_i x_i, \tag{39}$$

subject to:

$$\sum_{i=1}^{n} w_i x_i \leq W, \tag{40}$$

where v_i and w_i are the value and weight of item *i*, respectively, x_i is a binary variable indicating whether item *i* is included in the knapsack, and *W* is the weight capacity of the knapsack.

We now show how an instance of the Knapsack Problem can be reduced to our task offloading problem. Consider the following mapping: 1) Each task t_i in our problem corresponds to an item *i* in the Knapsack Problem. 2) The Energy cost E_{te} associated with offloading or executing task t_i corresponds to the weight w_i of item *i*. 3) The processing time $\mathcal{T}_i(t_{1:m})$ and transmission time T_s can be mapped to constraints similar to the weight capacity in the Knapsack Problem.

Thus, solving the task offloading problem involves determining which tasks to offload (analogous to selecting items to pack in the knapsack) to minimize the total energy consumption while satisfying the time constraints.

Given that the Knapsack Problem is NP-hard, and we have shown that our task offloading problem can be reduced from the Knapsack Problem, it follows that our problem is also NP-hard. This reduction implies that solving the task offloading problem in polynomial time would allow us to solve the Knapsack Problem in polynomial time, which is not possible unless P = NP. Therefore, the optimization problem defined by (11) is NP-hard. This conclusion justifies the use of heuristic or approximation algorithms to find feasible solutions within a reasonable time frame.

References

[1] H. Wang et al., "Low-complexity and efficient dependent subtask offloading strategy in IoT integrated with multi-access edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 1, pp. 621–636, Feb. 2024.

[2] X. Bi, X. Sun, Z. Lyu, B. Zhang, and X. Wei, "A back adjustment based dependent task offloading scheduling algorithm with fairness constraints in VEC networks," *Comput. Netw.*, vol. 223, Mar. 2023, Art. no. 109552.

[3] J. Zhang et al., "Dependent task offloading mechanism for cloud–edge-device collaboration," *J. Netw. Comput. Appl.*, vol. 216, Jul. 2023, Art. no. 103656.

[4] M. Liwang, Z. Gao, S. Hosseinalipour, Y. Su, X. Wang, and H. Dai, "Graph-represented computation-intensive task scheduling over air-ground integrated vehicular networks," *IEEE Trans. Service Comput.*, vol. 16, no. 5, pp. 3397–3411, Sep./Oct. 2023.

[5] B. Xu, Z. Kuang, J. Gao, L. Zhao, and C. Wu, "Joint offloading decision and trajectory design for UAV-enabled edge computing with task dependency," *IEEE Trans. Wireless Commun.*, vol. 22, no. 8, pp. 5043–5055, Aug. 2023.

[6] Z. Liu, M. Liwang, S. Hosseinalipour, H. Dai, Z. Gao, and L. Huang, "RFID: Towards low latency and reliable DAG task scheduling over dynamic vehicular clouds," *IEEE Trans. Veh. Technol.*, vol. 72, no. 9, pp. 12139–12153, Sep. 2023.

[7] L. X. Nguyen, Y. K. Tun, T. N. Dang, Y. M. Park, Z. Han, and C. S. Hong, "Dependency tasks offloading and communication resource allocation in collaborative UAVs networks: A metaheuristic approach," *IEEE Internet Things J.*, vol. 10, no. 10, pp. 9062–9076, May 2023.

[8] N. Sharma, A. Ghosh, R. Misra, and S. K. Das, "Deep meta Q-learning based multi-task offloading in edge-cloud systems," *IEEE Trans. Mobile Comput.*, vol. 23, no. 4, pp. 2583–2598, Apr. 2024.

[9] H. Yuan, Q. Hu, M. Wang, J. Bi, and M. Zhou, "Cost-minimized user association and partial offloading for dependent tasks in hybrid cloud–edge systems," in *Proc. IEEE Int. Conf. Autom. Sci. Eng. (CASE)*, 2022, pp. 1059–1064.

[10] R. Zhang et al., "OSTTD: Offloading of splittable tasks with topological dependence in multi-tier computing networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 555–568, Feb. 2023.

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

[12] X. Liu, S. Jiang, and Y. Wu, "A novel deep reinforcement learning approach for task offloading in MEC systems," *Appl. Sci.*, vol. 12, no. 21, 2022, Art. no. 11260.

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

[13] G. Liu, F. Dai, B. Huang, Z. Qiang, S. Wang, and L. Li, "A collaborative computation and dependency-aware task offloading method for vehicular edge computing: A reinforcement learning approach," *J. Cloud Comput.*, vol. 11 no. 1, p. 68, 2022.
[14] B. Gong and X. Jiang, "Dependent task-offloading strategy based on deep reinforcement learning in mobile edge computing," *Wireless Commun. Mobile Comput.*, vol. 2023, Jan. 2023, Art. no. 4665067.
[15] J. Wang, J. Hu, G. Min, W. Zhan, A. Y. Zomaya, and N. Georgalas, "Dependent task offloading for edge computing based on deep reinforcement learning," *IEEE Trans. Comput.*, vol. 71, no. 10, pp. 2449–2461, Oct. 2022.
[16] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Conf. Neur. Inf. Process. Syst.*, vol. 30, 2017, pp. 1–15.
[17] F. Liu, Z. Huang, and L. Wang, "Energy-efficient collaborative task computation offloading in cloud-assisted edge computing for IoT sensors," *Sensors.*, vol. 19, no. 5, p. 1105, 2019.
[18] Y. Sahni, J. Cao, L. Yang, and Y. Ji, "Multihop offloading of multiple DAG tasks in collaborative edge computing," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4893–4905, Mar. 2021.
[19] S. K. Roy, R. Devaraj, and A. Sarkar, "SAFLA: Scheduling multiple real-time periodic task graphs on heterogeneous systems," *IEEE Trans. Comput.*, vol. 72, no. 4, pp. 1067–1080, Apr. 2023.
[20] J. Liu, Y. Zhang, J. Ren, and Y. Zhang, "Auction-based dependent task offloading for IoT users in edge clouds," *IEEE Internet Things J.*, vol. 10, no. 6, pp. 4907–4921, Mar. 2023.
[21] J. Shi, J. Du, J. Wang, and J. Yuan, "Deep reinforcement learning-based V2V partial computation offloading in vehicular fog computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2021, pp. 1–6.
[22] S. Yun, M. Jeong, R. Kim, J. Kang, and H. J. Kim, "Graph transformer networks," in *Proc. 33rd Conf. Neur. Inf. Process. Syst.*, vol. 32, 2019, pp. 1–11.
[23] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jun. 2014, pp. 387–395.
[24] H. Zhou, K. Jiang, X. Liu, X. Li, and V. C. M. Leung, "Deep reinforcement learning for energy-efficient computation offloading in mobile-edge computing," *IEEE Internet Things J.*, vol. 9, no. 2, pp. 1517–1530, Jan. 2022.
[25] F. Liu, J. Huang, and X. Wang, "Joint task offloading and resource allocation for device-edge-cloud collaboration with subtask dependencies," *IEEE Trans. Cloud Comput.*, vol. 11, no. 3, pp. 3027–3039, Jul.–Sep. 2023.
[26] X. Dai et al., "Offloading dependent tasks in edge computing with unknown system-side information," *IEEE Trans. Service Comput.*, vol. 16, no. 6, pp. 4345–4359, Nov./Dec. 2023.
[27] U. Awada, J. Zhang, S. Chen, S. Li, and S. Yang, "Resource-aware multi-task offloading and dependency-aware scheduling for integrated edge-enabled IoV," *J. Syst. Archit.*, vol. 141, Aug. 2023, Art. no. 102923.
[28] J. Fang, D. Qu, H. Chen, and Y. Liu, "Dependency-aware dynamic task offloading based on deep reinforcement learning in mobile edge computing," *IEEE Trans. Netw. Service Manag.*, vol. 21, no. 2, pp. 1403–1415, Apr. 2024.
[29] L. Zhao et al., "MESON: A mobility-aware dependent task offloading scheme for urban vehicular edge computing," *IEEE Trans. Mobile Comput.*, vol. 23, no. 5, pp. 4259–4272, May 2024.
[30] S. Liu et al., "Dependent task scheduling and offloading for minimizing deadline violation ratio in mobile edge computing networks," *IEEE J. Sel. Areas Commun.*, vol. 41, no. 2, pp. 538–554, Feb. 2023.
[31] Z. Cao and X. Deng, "Dependent task offloading in edge computing using GNN and deep reinforcement learning," 2023, *arXiv:2303.17100*.
[32] C.-T. Mo, J.-H. Chen, and W. Liao, "Graph convolutional network augmented deep reinforcement learning for dependent task offloading in mobile edge computing," in *Proc. IEEE Wireless Commun. Netw. Conf. (WCNC)*, 2023, pp. 1–6.
[33] L. Zhang et al., "SoC-cluster as an edge server: An application-driven measurement study," 2022, *arXiv:2212.12842*.
[34] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[35] J. Achiam. "Spinning up in deep reinforcement learning." 2018. [Online]. Available: https://spinningup.openai.com/en/latest/

Bo Xie received the M.S. degree in computer technology from Guizhou University, Guiyang, China, in 2022. He is currently pursuing the Ph.D. degree in electronic science and technology from South China Normal University, China. His current research interests include mobile edge computing and optimization, and edge intelligence.

Haixia Cui (Senior Member, IEEE) received the M.S. and Ph.D. degrees in communication engineering from the South China University of Technology, Guangzhou, China, in 2005 and 2011, respectively. She is currently a Full Professor with the School of Electronic Science and Engineering, South China Normal University, China. From July 2014 to July 2015, she was an Advanced Visiting Scholar (Visiting Associate Professor) with the Department of Electrical and Computer Engineering, University of British Columbia, Vancouver, Canada. She has authored or coauthored more than 90 refereed journal and conference papers and 1 books. She also holds about 30 patents. Her current research interests are in the areas of mobile edge computing, vehicular networks, cooperative communication, wireless resource allocation, 5G/6G, multiple access control, and power control in wireless networks.

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.

Yejun He (Senior Member, IEEE) received the Ph.D. degree in information and communication engineering from the Huazhong University of Science and Technology, Wuhan, China, in 2005.

From 2005 to 2006, he was a Research Associate with the Department of Electronic and Information Engineering, The Hong Kong Polytechnic University, Hong Kong. From 2006 to 2007, he was a Research Associate with the Department of Electronic Engineering, Faculty of Engineering, The Chinese University of Hong Kong, Hong Kong. In 2012, he joined as a Visiting Professor with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada. From 2013 to 2015, he was an Advanced Visiting Scholar (Visiting Professor) with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA, USA. From 2023 to 2024, he is an Advanced Research Scholar (Visiting Professor) with the Department of Electrical and Computer Engineering, National University of Singapore, Singapore. Since 2006, he has been a Faculty with Shenzhen University, where he is currently a Full Professor with the College of Electronics and Information Engineering. He is also the Director of Sino-British Antennas and Propagation Joint Laboratory, Ministry of Science and Technology, China, Guangdong Engineering Research Center of Base Station Antennas and Propagation, and also with the Shenzhen Key Laboratory of Antennas and Propagation, and the Chair of IEEE Antennas and Propagation Society-Shenzhen Chapter. He was selected as a Leading Talent in the "Guangdong Special Support Program" and the Shenzhen "Pengcheng Scholar" Distinguished Professor, China, in 2024 and 2020, respectively. He has authored or coauthored more than 330 refereed journal and conference papers and seven books. He holds over 20 patents. He is the Principal Investigator for over 40 current or finished research projects, including the National Natural Science Foundation of China, the Science and Technology Program of Guangdong Province, and the Science and Technology Program of Shenzhen. His research interests include wireless communications, antennas, and radio frequency.

Dr. He was also a recipient of the Shenzhen Overseas High-Caliber Personnel Level B (Peacock Plan Award B) and Shenzhen High-Level Professional Talent (Local Leading Talent), the Second Prize of Guangdong Provincial Science and Technology Progress Award in 2023, the 10th Guangdong Provincial Patent Excellence Award in 2023, the Second Prize of Shenzhen Science and Technology Progress Award in 2017, and the Three Prize of Guangdong Provincial Science and Technology Progress Award in 2018, and the 2022 IEEE APS Outstanding Chapter Award. He is also an Associate Editor of the IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE ANTENNAS AND PROPAGATION MAGAZINE, IEEE ANTENNAS AND WIRELESS PROPAGATION LETTERS, *International Journal of Communication Systems*, *China Communications*, and *ZTE Communications*. He has served as a Reviewer for various journals, such as the IEEE TRANSACTIONS ON VEHICULAR TECHNOLOGY, the IEEE TRANSACTIONS ON COMMUNICATIONS, the IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, the IEEE TRANSACTIONS ON ANTENNAS AND PROPAGATION, IEEE WIRELESS COMMUNICATIONS, IEEE COMMUNICATIONS LETTERS, *International Journal of Communication Systems*, and *Wireless Personal Communications*. He was a Technical Program Committee Member or a Session Chair for various conferences, including the IEEE Global Telecommunications Conference, IEEE International Conference on Communications, IEEE Wireless Communication Networking Conference, and IEEE Vehicular Technology Conference. He was the TPC Chair for IEEE ComComAp 2021 and the General Chair for IEEE ComComAp 2019. He was selected as a Board Member of the IEEE Wireless and Optical Communications Conference. He was the TPC Co-Chair for WOCC 2015, 2019, 2022, and 2023, respectively, APCAP 2023, UCMMT 2023, ACES-China2023, and NEMO 2020. He was an Executive Chair of 2024 IEEE International Workshop of Radio Frequency and Antenna Technologies in 2024. He is also an Executive Chair of 2025 IEEE International Workshop of Radio Frequency and Antenna Technologies in 2025. He acted as the Publicity Chair of several international conferences such as the IEEE PIMRC 2012, and IEEE MAPE 2024. He is a Fellow of the China Institute of Communications and IET.

Mohsen Guizani (Fellow, IEEE) received the B.S. (Distinction), M.S., and Ph.D. degrees in electrical and computer engineering from Syracuse University, Syracuse, NY, USA, in 1985, 1987, and 1990, respectively. He is currently a Professor of machine learning with the Mohamed Bin Zayed University of Artificial Intelligence, Abu Dhabi, UAE. Previously, he worked in different institutions in USA. He is the author of 11 books, more than 1000 publications and several U.S. patents. His research interests include applied machine learning and artificial intelligence, smart city, Internet of Things, intelligent autonomous systems, and cyber-security. He has won several research awards including the "2015 IEEE Communications Society Best Survey Paper Award", the Best ComSoc Journal Paper Award in 2021 and the 5 Best Paper Awards from ICC and Globecom Conferences. He is also the recipient of the 2017 IEEE Communications Society Wireless Technical Committee Recognition Award, the 2018 AdHoc Technical Committee Recognition Award, and the 2019 IEEE Communications and Information Security Technical Recognition Award. He was listed as a Clarivate Analytics Highly Cited Researcher in Computer Science from 2019 to 2022. He served as an Editor-in-Chief of IEEE NETWORK and is currently serving an Editorial Boards of many the IEEE TRANSACTIONS AND MAGAZINES. He was the Chair of the IEEE Communications Society Wireless Technical Committee and TAOS Technical Committee. He served as the IEEE Computer Society Distinguished Speaker and is currently the IEEE ComSoc Distinguished Lecturer.

Authorized licensed use limited to: SHENZHEN UNIVERSITY. Downloaded on November 30,2025 at 01:59:41 UTC from IEEE Xplore. Restrictions apply.