This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

Neural Combinatorial Optimization for Multiobjective Task Offloading in Mobile Edge Computing

Xiang-Jie Xiao, Yong Wang, *Senior Member, IEEE*, Pei-Qiu Huang, *Member, IEEE*, and Kezhi Wang, *Senior Member, IEEE*

***Abstract*—Task offloading is crucial in supporting resource-intensive applications in mobile edge computing. This paper explores multiobjective task offloading, aiming to minimize energy consumption and latency simultaneously. Although learning-based algorithms have been used to address this problem, they train a model based on one *a priori* preference to make the offloading decision. When the preference changes, the trained model may not perform well and needs to be retrained. To address this issue, we propose a neural combinatorial optimization method that combines an encoder-decoder model with reinforcement learning. The encoder captures task relationships, while the decoder, equipped with a preference-based attention mechanism, determines offloading decisions for various preferences. Additionally, reinforcement learning is employed to train the encoder-decoder model. Since the proposed method can infer the offloading decision for each preference, it eliminates the need to retrain the model when the preference changes, thus improving real-time performance. Experimental studies demonstrate the effectiveness of the proposed method by comparison with three algorithms on instances of different scales.**

***Index Terms*—Mobile edge computing, task offloading, multiobjective, neural combinatorial optimization, encoder-decoder model**

I. Introduction

The growing popularity of new applications, such as online gaming [1], health monitoring [2], and online video streaming [3], has resulted in a substantial increase in the demand for computing and storage resources. Although the computing power and storage capacity of mobile devices have improved recently, the portability requirements of mobile devices still necessitate designers to balance factors including computing power, battery life, and device size, making it challenging to meet the execution demands of such applications in the foreseeable future [4]. Mobile edge computing (MEC) has emerged as a critical technology to address these demands by deploying edge nodes equipped with storage and computing resources close to the edge of the network [5]–[7]. By offloading tasks to edge nodes, energy consumption and latency on user equipment (UE) may be reduced.

This work was supported in part by the National Natural Science Foundation of China under Grant U23A20347 and in part by the Royal Society International Exchange project under Grant IEC-NSFC-211264. *(Corresponding authors: Yong Wang; Pei-Qiu Huang)*

X.-J. Xiao, Y. Wang, and P.-Q. Huang are with the School of Automation, Central South University, Changsha 410083, China (e-mail: xjxiao@csu.edu.cn; ywang@csu.edu.cn; pqhuang@csu.edu.cn).

K. Wang is with the Department of Computer Science, Brunel University London, Uxbridge, Middlesex UB8 3PH, UK. (e-mail: kezhi.wang@brunel.ac.uk).

Within the task offloading process, the offloading decision primarily determines which tasks to be offloaded, significantly influencing Quality of Service (QoS). Traditional methods, including mathematical programming (e.g., dynamic programming) and heuristic techniques (e.g., evolutionary algorithms) [8]–[10], face significant challenges in making real-time offloading decisions due to their significant computational burdens or the need for multiple iterations [11]. As a result, many learning-based algorithms have received considerable attention. For example, Huang *et al.* [12] developed a reinforcement learning technique using deep neural networks for task offloading in wireless-powered MEC systems. Gao *et al.* [13] proposed a multi-agent deep deterministic policy gradient algorithm to obtain the offloading decision for multiple wireless devices. Dai *et al.* [14] introduced an asynchronous deep Q-learning method for making the offloading decision in MEC-enhanced vehicular networks. Tan *et al.* [15] devised a decentralized offloading decision-making approach based on multi-agent reinforcement learning. Chai *et al.* [16] combined attention mechanisms with reinforcement learning for task offloading in unmanned aerial vehicle (UAV)-assisted MEC systems. Zhao *et al.* [17] investigated a cooperative multi-agent deep reinforcement learning approach to optimize the offloading decision in multi-UAV systems.

However, the aforementioned studies primarily focus on single-objective task offloading. In real-world scenarios, multiple objectives need to be considered [18]. Consequently, several studies have proposed various learning-based algorithms for multiobjective task offloading [19]–[21]. These studies generally account for multiple objectives, such as energy consumption, latency, and resource utilization, to balance multiple objectives to enhance overall system performance. While significant progress has been made, these studies typically train a model based on one *a priori* preference among these objectives (usually in the form of weights) [20]. However, when the preference changes, the performance of the trained model may significantly degrade, and it may be necessary to retrain the model. To address this issue, this paper proposes a neural combinatorial optimization method that utilizes an encoder-decoder model and reinforcement learning to learn an offloading decision policy. This policy can adapt to all potential preferences. As a result, the proposed method can directly make an offloading decision without requiring retraining the model in the case of preference changes. Hence, the proposed method exhibits high flexibility and real-time performance.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

The main contributions of this paper are summarized as follows.

- This paper studies multiobjective task offloading with the aim of minimizing both energy consumption and latency without the *a priori* preference.
- A neural combinatorial optimization method is developed, which combines the encoder-decoder model with reinforcement learning. The encoder captures the relevant information among tasks, while the preference-conditioned decoder makes the offloading decision for any given preference. Furthermore, reinforcement learning is employed to train the encoder-decoder model, making it applicable to all potential preferences.
- The proposed method is compared with three other existing methods on instances of various scales. The results show the superior performance of the proposed method.

The reminder of this paper is organized as follows. Section II introduces the related work. The system model and problem formulation are presented in Section III. Section IV describes the details of the proposed algorithm. The experimental studies are given in Section V. Finally, Section VI concludes this paper.

II. Related Work

Several studies have attempted to investigate multiobjective task offloading in MEC systems to optimize multiple objectives simultaneously. These studies mainly focus on evolutionary algorithms and learning-based algorithms [22]–[25].

Due to the population-based nature, evolutionary algorithms can provide a set of offloading decisions after a single run. For example, Bozorgchenani *et al.* [26] developed a multiobjective genetic algorithm for task offloading in a two-tier MEC system to minimize energy consumption and processing delay. Li *et al.* [27] jointly optimized the offloading decision and the cache placement via a multiobjective artificial bee colony algorithm to maximize the hit ratio and minimize the service latency. Xiao *et al.* [28] proposed a binary particle swarm optimization algorithm to determine the optimal content caching and parallel offloading decision for minimizing delay and energy consumption. Ma *et al.* [29] utilized a particle swarm optimization method to determine the optimal offloading decisions for dependent tasks, aiming to minimize both the completion time and the execution cost. The aforementioned methods can obtain a set of offloading decisions accommodated to multiple preferences. However, they encounter two challenges in making real-time offloading decisions. First, multiple iterations are required to obtain the offloading decisions. Second, practitioners must further choose a suitable offloading decision from the available alternatives based on their preference.

To achieve better real-time performance, learning-based algorithms are utilized for multiobjective task offloading in MEC systems since they can directly infer the offloading decision. For instance, Yan *et al.* [19] developed a deep reinforcement learning approach for joint task offloading and resource allocation in MEC systems to minimize the weighted sum of total energy consumption and execution time. Wang *et al.* [20] proposed a novel neural combinatorial optimization method to optimize the offloading decision with the aim of minimizing the weighted sum of delay and energy consumption. Zhang *et al.* [21] utilized a double deep Q-network to optimize multiple objectives related to computational load, delay, and energy consumption. Subsequently, they performed the Chebyshev scalarization for multiple Q values. Such algorithms transform multiple objectives into a scalar objective via one *a priori* preference. Their effectiveness may be limited to the *a priori* preference and their performance may degenerate when the preference changes. Considering that the preference is challenging to preset and may vary dynamically, this paper proposes a multiobjective task offloading method that can accommodate all potential preferences. A comparison between our work and the current work is summarized in Table I.

III. System Model and Problem Formulation

Nomenclature

λ	The preference between energy consumption and latency
$\boldsymbol{a} = [a_1, \cdots, a_n]$	Offloading decisions of n tasks
AT_i^d	The time at which the downlink channel becomes idle in preparation to receive the results of task i
AT_i^e	The time at which the MEC server becomes idle in preparation to execute task i
AT_i^l	The time at which the local processor becomes idle in preparation to execute task i
AT_i^u	The time at which the uplink channel becomes idle in preparation to transmit task i to the MEC server
C_i	The total number of CPU cycles required by task i
D_i^d	The size of the results of task i
D_i^u	The size of task i sent to the MEC server for processing
E	The total energy consumed by the UE for executing all tasks
E_i^l	The energy consumption of the local execution for executing task i locally
E_i^r	The energy consumption of the remote execution for task i
f^e	The allocated computation capability by the MEC server
f^l	The computation capability of the local processor
FT	The total latency for executing all tasks
FT_i^d	The finishing time of receiving the results of task i
FT_i^e	The finishing time of executing task i at the MEC server
FT_i^l	The finishing time of task i on the local processor
FT_i^u	The finishing time of transmitting task i to the MEC server
n	Number of Tasks
P^d	The power of the UE for receiving the results of tasks
P^u	The power of the UE for sending the tasks to the MEC server
R^d	The downlink transmission rate
R^u	The uplink transmission rate
T_i^d	The transmission time of task i on the downlink channel

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

TABLE I
A COMPARISON BETWEEN OUR WORK AND THE CURRENT WORK.

Method	Reference	Objective functions	Real-time performance	Number of accommodated preferences
Evolutionary algorithm	[26]	Energy consumption, processing delay	Low	Multiple preferences
	[27]	Hit ratio, service latency	Low	Multiple preferences
	[28]	Delay, energy consumption	Low	Multiple preferences
	[29]	Completion time, execution cost	Low	Multiple preferences
Learning-based algorithm	[19]	Energy consumption, execution time	High	One *a priori* preference
	[20]	Latency, energy consumption	High	One *a priori* preference
	[21]	Computational load, delay, energy consumption	High	One *a priori* preference
	Our work	Latency, energy consumption	High	All potential preferences

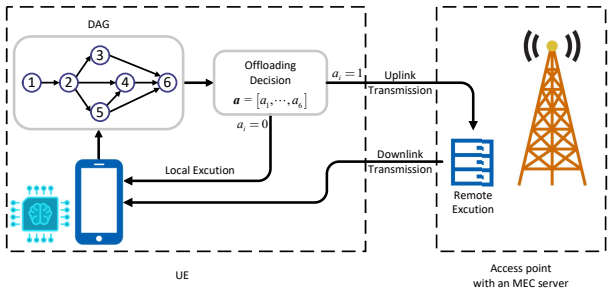

Fig. 1. An MEC system consisting of a UE and an access point with an MEC server.

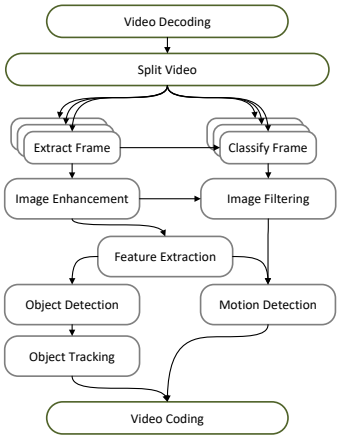

Fig. 2. An example of DAG for facial recognition application [30].

T_i^e The execution time of task i on the MEC server
T_i^u The transmission time of task i on the uplink channel
T_i^l The execution time of task i on the local processor

A. System Model

As depicted in Fig. 1, this paper investigates an MEC system consisting of a UE and an access point with an MEC server. The UE hosts an application that consists of multiple dependent tasks. Such applications are widespread, as illustrated in Fig. 2, one notable example being the facial recognition application [30]. These tasks in the applications can be modeled as a Directed Acyclic Graph (DAG). We use $G = (\mathcal{T}, \mathcal{E})$ to represent the DAG, where $\mathcal{T}$ is the set of vertices representing tasks and $\mathcal{E}$ is the set of directed edges representing task dependencies. Vertex $i \in \mathcal{T}$ represents task i and directed edge $e(i,j) \in \mathcal{E}$ represents the dependency between tasks i and j. A task can only be executed once its predecessors have been completed, and a terminal task has no successors. For example, there are six tasks in Fig. 3, i.e., tasks 1 to 6, that need to be executed. After task 2 is completed, tasks 3 and 5 can be executed. Task 4 is dependent on the completion of tasks 2 and 5. Similarly, task 6 cannot be executed until tasks 3 to 5 have all been completed.

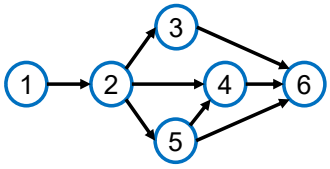

Fig. 3. A DAG consisting of tasks 1 to 6.

Task $i \in \mathcal{T}$ is represented by a three-tuple (C_i, D_i^u, D_i^d), where C_i represents the required number of CPU cycles, D_i^u represents the amount of data offloaded to the MEC server for processing, and D_i^d represents the size of the received result. We assume that each task i can be executed either on the local processor of the UE (i.e., local execution) or on the MEC server (i.e., remote execution). The offloading decision is denoted by $\boldsymbol{a} = [a_1, \ldots, a_n]$, where n denotes the task number, and $a_i = 0$ and $a_i = 1$ indicate the local and remote execution of task i, respectively. For local execution, task i is executed directly on the local processor. In contrast, in the case where task i is offloaded to the MEC server for remote execution, the process involves three stages: 1) the UE sends task i to the MEC server via the uplink transmission; 2) the MEC server executes task i; and 3) the UE receives the results of task i from the MEC server via the downlink transmission.

Moreover, we define the finishing time of executing task i on the local processor, sending task i to the MEC server, executing task i on the MEC server, and transmitting the results of task i back to the UE as FT_i^l, FT_i^u, FT_i^e, and FT_i^d, respectively.

1) Local Execution: The execution time of task i on the local processor is given by

$$T_i^l = \frac{C_i}{f^l} \tag{1}$$

where f^l denotes the computation capability (i.e., CPU cycles per second) of the local processor.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

To execute task i locally, two conditions must be fulfilled: 1) the local processor must be idle, as it cannot process multiple tasks concurrently; 2) task dependencies require that all predecessor tasks (denoted as $\mathcal{P}(i)$) of task i are completed either locally or remotely [20]. Considering condition 1), the available time at which the local processor becomes idle in preparation to execute task i is given by

$$AT_i^l = \max\{AT_j^l, FT_j^l\} \tag{2}$$

where task j is executed prior to task i. Note that, if task j is not executed locally, $FT_j^l = 0$. Then, further considering condition 2), the finishing time of executing task i on the local processor is given by

$$FT_i^l = \max_{k\in\mathcal{P}(i)} \{AT_i^l, FT_k^l, FT_k^d\} + T_i^l. \tag{3}$$

For task i in the local execution, we only need to consider the energy consumption of the local processor as follows

$$E_i^l = c(f^l)^\beta T_i^l \tag{4}$$

where $c > 0$ is a constant that depends on the average switched capacitance and average activity factor, and $\beta \geq 2$ is a constant.

2) Remote Execution: The time spent on uplink and downlink transmissions of task i is calculated as

$$T_i^u = \frac{D_i^u}{R^u} \tag{5}$$

and

$$T_i^d = \frac{D_i^d}{R^d} \tag{6}$$

where R^u and R^d represent the transmission rate for uplink and downlink transmissions, respectively. The execution time of task i on the MEC server is expressed as

$$T_i^e = \frac{C_i}{f^e} \tag{7}$$

where f^e represents the computation capability of the MEC server allocated to task i.

It is noteworthy that both the uplink and downlink channels, as well as the MEC server, have the capacity to handle only a single task at any given time. Therefore, the available time at which the uplink channel becomes idle in preparation to transmit task i to the MEC server can be calculated as

$$AT_i^u = \max\{AT_j^u, FT_j^u\} \tag{8}$$

and the available time at which the MEC server becomes idle in preparation to execute task i is expressed as

$$AT_i^e = \max\{AT_j^e, FT_j^e\}. \tag{9}$$

Similarly, the available time at which the downlink channel becomes idle in preparation to receive the results of task i is given by

$$AT_i^d = \max\{AT_j^d, FT_j^d\} \tag{10}$$

where task j is executed prior to task i. Note that, $FT_j^u = 0$, $FT_j^e = 0$, and $FT_j^d = 0$ if task j is executed locally.

TABLE II
AVAILABLE AND FINISHING TIME FOR TASKS 1 TO 6.

Task	T_i^l	T_i^u	T_i^e	T_i^d	AT_i^l	AT_i^u	AT_i^e	AT_i^d	FT_i^l	FT_i^u	FT_i^e	FT_i^d
1	2	2	1	1	0	0	0	0	2	0	0	0
2	4	1	1	1	2	0	0	0	0	3	4	5
3	3	1	1	1	2	3	4	5	8	0	0	0
4	5	3	2	1	8	3	4	5	0	8	10	11
5	2	1	1	1	8	8	10	11	0	9	11	12
6	4	2	3	2	8	9	11	12	16	0	0	0

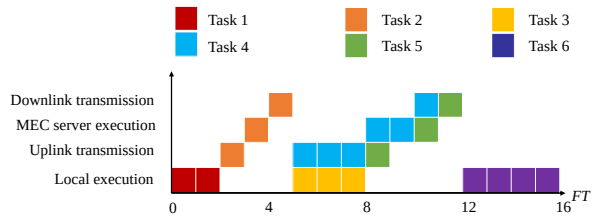

Fig. 4. A Gantt chart corresponding to Table II.

Since the task dependencies require that all predecessor tasks in $\mathcal{P}(i)$ are completed either locally or remotely, the finishing time of sending task i to the MEC server is given by

$$FT_i^u = \max_{k\in\mathcal{P}(i)} \{AT_i^u, FT_k^l, FT_k^d\} + T_i^u. \tag{11}$$

Similarly, the finishing time of executing task i on the MEC server is

$$FT_i^e = \max\{AT_i^e, FT_i^u\} + T_i^e. \tag{12}$$

Finally, we calculate the finishing time of receiving the results of task i as

$$FT_i^d = \max\{AT_i^d, FT_i^e\} + T_i^d. \tag{13}$$

For task i in the remote execution, we only need to consider the energy consumption of the UE, including the energy consumption for sending task i and receiving the results:

$$E_i^r = P^u T_i^u + P^d T_i^d \tag{14}$$

where P^u and P^d represent the power of sending task i and receiving the results, respectively.

Table II presents an example of calculating the available and finishing time for tasks 1 to 6 in Fig. 3. Herein, we assume that the values of T_i^l, T_i^u, T_i^e, and T_i^d of tasks 1 to 6 are known (i.e., columns 2 to 5 in Table II). In addition, we also assume that tasks 1 to 6 make the offloading decision sequentially and their offloading decision is $\boldsymbol{a} = [0, 1, 0, 1, 1, 0]$. Based on (2), (3), and (8)-(13), we can calculate the available and finishing time of tasks 1 to 6 (i.e., columns 6 to 13 in Table II). The Gantt chart in Fig. 4 intuitively demonstrates the execution process of the tasks.

B. Problem Formulation

In this paper, we aim to optimize the offloading decision (i.e., $\boldsymbol{a}$) to reduce the energy consumption and the latency of executing all tasks. The energy consumption of the UE for executing all tasks is expressed as:

$$E(\boldsymbol{a}) = \sum_{i=1}^{n} ((1 - a_i)E_i^l + a_i E_i^r). \tag{15}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

Moreover, the latency of executing all tasks is given by

$$FT(\boldsymbol{a}) = \max_{i=1,\dots,n} \{(1-a_i)FT_i^l + a_i FT_i^d\}. \tag{16}$$

Therefore, the multiobjective task offloading problem can be expressed as follows

$$\min_{\boldsymbol{a}} \begin{cases} E(\boldsymbol{a}) = \sum_{i=1}^{n}((1-a_i)E_i^l + a_i E_i^r) \\ FT(\boldsymbol{a}) = \max_{i=1,\dots,n}\{(1-a_i)FT_i^l + a_i FT_i^d\} \end{cases}$$
$$\text{s.t.} \quad a_i \in \{0,1\}, i = 1, \cdots, n. \tag{17}$$

In practice, the weighted sum method is widely used to combine all the multiple objectives into a scalar objective [31]. Therefore, we can obtain the following QoS metric by calculating the weighted sum of the differences between the offloading decision and the local execution in terms of normalized energy consumption and latency of executing all tasks [20], [32]:

$$\max_{\boldsymbol{a}} J(\boldsymbol{a}|\lambda) = \lambda \frac{E^l - E(\boldsymbol{a})}{E^l} + (1-\lambda)\frac{FT^l - FT(\boldsymbol{a})}{FT^l} \tag{18}$$

where E^l represents the energy consumption of executing all tasks locally, and FT^l represents the latency of executing all tasks locally. The weight λ represents the preference between energy consumption and latency.

Note that, λ may vary in different situations. For mobile devices such as smartphones and laptops, user preferences for performance metrics, including latency and energy consumption, can vary significantly based on the battery status. When the battery is fully charged or near full capacity, users prefer low latency because applications such as online gaming, video streaming, or real-time communication require low latency to ensure seamless interaction. Conversely, as the battery level drops, users may prefer to reduce energy consumption, even if it results in slightly higher latency, to ensure the mobile devices remain operational for as long as possible. Because λ varies dynamically, there is no unique optimal offloading decision in (18), and the optimal offloading decision may vary depending on λ.

Remark 1: For ease of reference, the key notations used in this section are summarized at the beginning of this section.

IV. Proposed Algorithm

A. General Framework

The optimization problem in (18) is a typical combinatorial optimization problem. Neural combinatorial optimization, as a learning-based method, can leverage deep learning to automatically learn effective methods for combinatorial optimization problems [33], [34], and has consequently been applied to tackle (18) in [20]. It is worth noting that [20] only trains a model based on one *a priori* preference, and if the preference changes, the model may not perform well. Under this condition, the model may need to be retrained, potentially resulting in inefficiency. Inspired by the recently proposed Pareto set learning [35], [36], we design a novel neural combinatorial optimization method, called DepTaskNet, to solve (18). The method does not rely on the *a priori* preference and can adapt to accommodate any potential preference.

The primary process of DepTaskNet is outlined as follows. DepTaskNet first sorts the tasks in the DAG and generates the input features for each task. The encoder then extracts the relevant information among the sorted tasks based on their input features. Finally, the decoder makes the offloading decision for each sorted task sequentially based on the output of the encoder and the preference. In addition, the encoder-decoder model is trained via reinforcement learning, with the aim of maximizing $J(\boldsymbol{a}|\lambda)$ in (18) for each randomly sampled λ. In this way, DepTaskNet is able to accommodate all potential preferences and directly infer the offloading decision when the preference changes, alleviating the need for model retraining. Fig. 5 presents an example of DepTaskNet to make the offloading decision for a DAG consisting of tasks 1 to 6. The following describes the proposed DepTaskNet in detail.

B. Task Sorting

Due to the task dependencies, we must make the offloading decision for each task sequentially. As a result, before making the offloading decision, it is critical to organize the tasks in the DAG in a specific order. DepTaskNet adopts the method in [20] to sort the tasks in descending order according to the following rank value:

$$r(i) = \begin{cases} T_i^o, & \text{if task } i \text{ is the terminal task} \\ \max_{j\in\mathcal{S}(i)} r(j) + T_i^o, & \text{otherwise} \end{cases} \tag{19}$$

where $T_i^o = \min\{T_i^l, T_i^u + T_i^e + T_i^d\}$ ($i \in \mathcal{T}$) and $\mathcal{S}(i)$ denotes the set of immediate successors for task i in the DAG. Then, a task sequence $p_1, \dots, p_n$ can be obtained based on $r(i)$, where p_j ($j \in 1, \dots, n$) represents the task index of the jth task after sorting. Next, we sequentially make the offloading decision for each task p_j. This ensures that the offloading decision for the current task in the DAG is always determined before its subsequent tasks, thus satisfying task dependencies.

After that, it is imperative to develop more effective strategies to enhance the capture of task-related information and dependencies among tasks. We generate an input feature $\boldsymbol{x}_i$ for each task $i \in \mathcal{T}$, which includes: 1) the task index of task i, 2) the values of T_i^l, T_i^u, T_i^e, and T_i^d for task i, and 3) the task indexes of μ tasks in $\mathcal{S}(i)$ and μ tasks in $\mathcal{P}(i)$. If the number of tasks in $\mathcal{S}(i)$ or $\mathcal{P}(i)$ is no more than μ, we supplement the task indexes with -1.

C. Encoder

Capturing the underlying information among the sorted tasks is beneficial to making the offloading decision. To achieve this, the encoder is employed to generate embeddings of all tasks. These embeddings can be regarded as crucial features extracted from the context information of tasks. The encoder in DepTaskNet consists of two layers: a node wise fully connected feed-forward (FF) layer and a bi-directional long short-term memory (BiLSTM) layer that executes bi-directional message passing among the tasks.

First, the FF layer projects each $\boldsymbol{x}_{p_j}$ into a 128-dimension vector. Then, the outputs of the FF layer (i.e., $\boldsymbol{h}_{p_1}^{ef}, \dots, \boldsymbol{h}_{p_n}^{ef}$) are fed into the BiLSTM layer. Unlike the commonly used long short-term memory (LSTM) network that solely relies

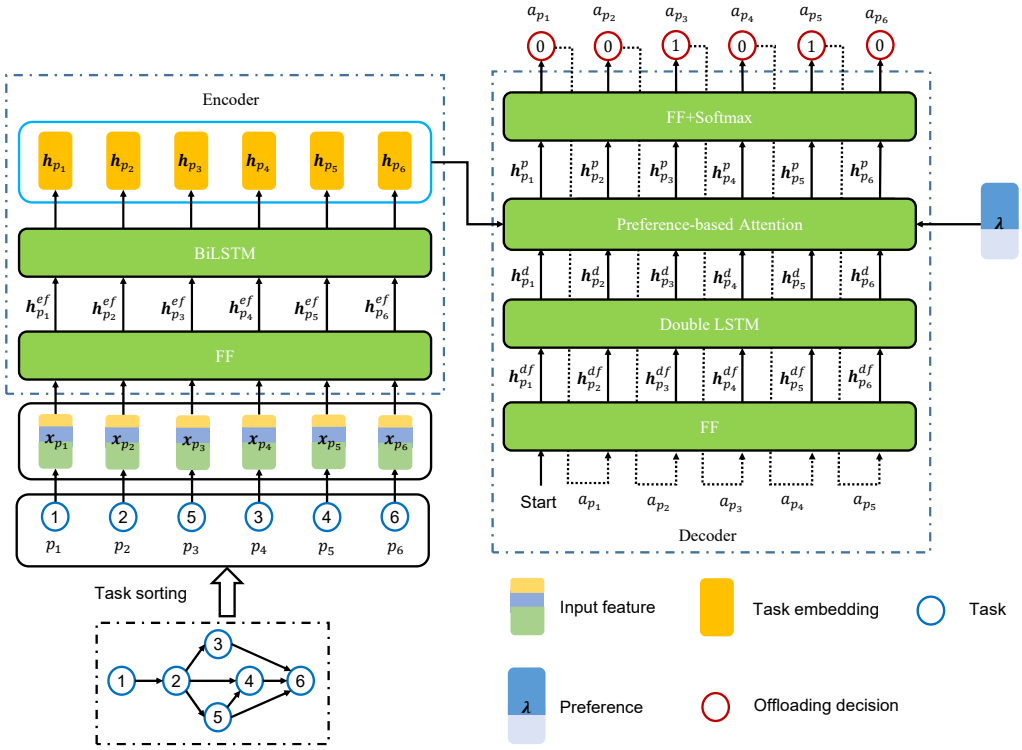

Fig. 5. An example of DepTaskNet to make the offloading decision for a DAG consisting of tasks 1 to 6.

on backward information, the BiLSTM leverages forward and backward information simultaneously; thus, the context information of tasks can be comprehensively captured [37]. In the BiLSTM layer, the output of the forward and backward processes for task p_j are denoted as $\boldsymbol{h}_{p_j}^f$ and $\boldsymbol{h}_{p_j}^b$, respectively [38]. Finally, the embedding of task p_j which is the output of the BiLSTM layer can be calculated as

$$\boldsymbol{h}_{p_j} = \sigma(\mathbf{W}^{hh}[\boldsymbol{h}_{p_j}^f; \boldsymbol{h}_{p_j}^b] + \boldsymbol{b}^{hh}) \tag{20}$$

where $\mathbf{W}^{hh}$ represents a trainable matrix, $\boldsymbol{b}^{hh}$ represents a trainable bias vector, and $[\boldsymbol{h}_{p_j}^f; \boldsymbol{h}_{p_j}^b]$ represents a concatenate operation that connects $\boldsymbol{h}_{p_j}^f$ and $\boldsymbol{h}_{p_j}^b$ as one vector.

D. Decoder

DepTaskNet then uses the decoder to make, in sequence, the offloading decision (i.e., a_{p_j}) for each sorted task p_j based on the embeddings (i.e., $\boldsymbol{h}_{p_1}, \ldots, \boldsymbol{h}_{p_n}$) of all tasks and the preference (i.e., λ). Since taking λ into account, the decoder can generate the corresponding offloading decision after λ changes. In addition, to enable the decoder to more effectively utilize $\boldsymbol{h}_{p_1}, \ldots, \boldsymbol{h}_{p_n}$, an attention mechanism is introduced to assign different weights to them. As shown in Fig. 5, the decoder includes four layers: the FF layer, the double LSTM layer, the preference-condition attention layer, and the FF layer with the softmax activation function. The detailed process of the decoder to obtain a_{p_j} for task p_j is as follows.

We first use the FF layer to project the offloading decision (i.e., $a_{p_{j-1}}$) of task p_{j-1} into a 128-dimensional vector $\boldsymbol{h}_{p_j}^{df} = \mathbf{W}^e a_{p_{j-1}}$, where $\mathbf{W}^e$ is a trainable matrix. Subsequently,

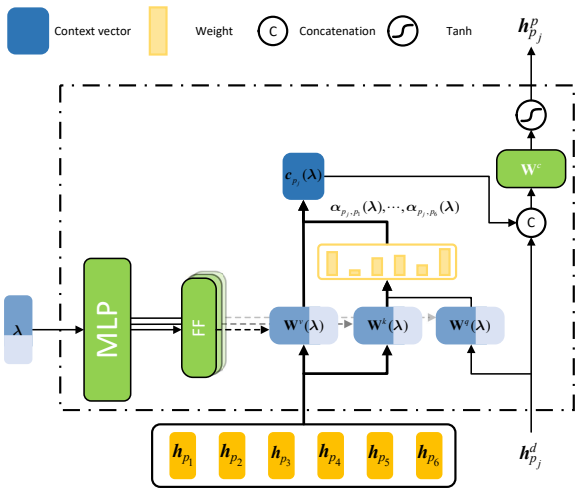

Fig. 6. Preference-condition attention layer.

$\boldsymbol{h}_{p_j}^{df}$ is fed into the double LSTM layer to obtain the hidden state $\boldsymbol{h}_{p_j}^d$ for task p_j. The double LSTM layer includes two LSTM networks, both of which perform the forward process to capture the backward information, and the output of the first LSTM network is the input of the second LSTM network.

Subsequently, as illustrated in Fig. 6, the preference-condition attention layer initially uses a multilayer perceptron model to generate three 2-dimensional vectors conditioned on λ and employs three FF layers to project these three vectors into three high-dimensional vectors as in [35], which are

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

subsequently reshaped into a $128 * 128$ matrix, a $128 * 256$ matrix, and a $256 * 256$ matrix (denoted as $\mathbf{W}^q(\lambda), \mathbf{W}^k(\lambda)$, and $\mathbf{W}^v(\lambda)$), respectively.

After that, we calculate the weight of each $\boldsymbol{h}_{p_i}(i = 1, \ldots, n)$ for $\boldsymbol{h}^d_{p_j}$:

$$\alpha_{p_j,p_i}(\lambda) = \frac{\exp(score(\boldsymbol{h}^d_{p_j}, \boldsymbol{h}_{p_i}|\lambda))}{\sum_{i=1}^{n} \exp(score(\boldsymbol{h}^d_{p_j}, \boldsymbol{h}_{p_i}|\lambda)}, i = 1, \ldots, n \tag{21}$$

where $score(\boldsymbol{h}^d_{p_j}, \boldsymbol{h}_{p_i}|\lambda)$ measures the alignment between $\boldsymbol{h}^d_{p_j}$ and each $\boldsymbol{h}_{p_i}$:

$$score(\boldsymbol{h}^d_{p_j}, \boldsymbol{h}_{p_i}|\lambda) = (\mathbf{W}^q(\lambda)\boldsymbol{h}^d_{p_j})^{\mathrm{T}}\mathbf{W}^k(\lambda)\boldsymbol{h}_{p_i}, i = 1, \ldots, n. \tag{22}$$

Next, based on $\alpha_{p_j,p_1}(\lambda), \ldots, \alpha_{p_j,p_n}(\lambda)$, the context vector (denoted as $c_{p_j}(\lambda)$) for task p_j is obtained via the weight sum of $\boldsymbol{h}_{p_1}, \ldots, \boldsymbol{h}_{p_n}$:

$$\boldsymbol{c}_{p_j}(\lambda) = \sum_{i=1}^{n} \alpha_{p_j,p_i}(\lambda)\mathbf{W}^v(\lambda)\boldsymbol{h}_{p_i}. \tag{23}$$

It can be observed from (21) that as the alignment between $\boldsymbol{h}^d_{p_j}$ and $\boldsymbol{h}_{p_i}$ increases, the value of α_{p_j,p_i} also increases, resulting in a higher utilization of $\boldsymbol{h}_{p_i}$. After that, we concatenate $c_{p_j}(\lambda)$ and $\boldsymbol{h}^d_{p_j}$, which are then projected to a 128-dimensional vector:

$$\boldsymbol{h}^p_{p_j} = \tanh(\mathbf{W}^c[\boldsymbol{c}_{p_j}(\lambda); \boldsymbol{h}^d_{p_j}]) \tag{24}$$

where $\mathbf{W}^c$ is a trainable matrix.

Finally, $\boldsymbol{h}^p_{p_j}$ is transformed into a 2-dimensional vector via a FF layer. This vector is then used to determine the probability of executing task p_j locally or remotely (denoted as $p_{\boldsymbol{\theta}(\lambda)}(a_{p_j}|G, a_{p_1}, \ldots, a_{p_{j-1}})$) via a softmax activation function, where $\boldsymbol{\theta}(\lambda)$ represents the overall trainable parameters of the encoder-decoder model conditioned on λ. Based on $p_{\boldsymbol{\theta}(\lambda)}(a_{p_j}|G, a_{p_1}, \ldots, a_{p_{j-1}})$, a_{p_j} is obtained for task p_j.

Remark 1: As shown in Fig. 5, we first sort tasks 1 to 6 based on Section IV-B. Suppose that the indexes of the sorted tasks are $p_1, \ldots, p_6$. The input features of tasks $p_1, \ldots, p_6$ are then generated as $\boldsymbol{x}_{p_1}, \ldots, \boldsymbol{x}_{p_6}$. Subsequently, as introduced in Section IV-C, the relevant information among tasks $p_1, \ldots, p_6$ (i.e., $\boldsymbol{h}_{p_1}, \ldots, \boldsymbol{h}_{p_6}$) is extracted via the encoder based on $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_6$. After that, we employ the decoder introduced in Section IV-D to sequentially make the offloading decision a_{p_j} for each task p_j $(j = 1, \ldots, 6)$ based on $\boldsymbol{h}_{p_1}, \ldots, \boldsymbol{h}_{p_6}$ and the preference λ.

E. Training Procedure

The encoder-decoder model defines an offloading decision policy $p_{\boldsymbol{\theta}(\lambda)}(\boldsymbol{a}|G)$ parameterized by $\boldsymbol{\theta}(\lambda)$ to make the offloading decision in sequence as follows

$$p_{\boldsymbol{\theta}(\lambda)}(\boldsymbol{a}|G) = \prod_{j=1}^{n} p_{\boldsymbol{\theta}(\lambda)}(a_{p_j}|G, a_{p_1}, \ldots, a_{p_{j-1}}). \tag{25}$$

Our training procedure aims to find the optimal offloading decision policy $p_{\boldsymbol{\theta}^*(\lambda)}(\boldsymbol{a}|G)$ that maximizes $J(\boldsymbol{a}|\lambda)$ for each randomly sampled λ. $p_{\boldsymbol{\theta}(\lambda)}(\boldsymbol{a}|G)$ can be trained using both supervised and unsupervised learning. However, due to the NP-hard nature of the studied problem, acquiring high-quality labels required in supervised learning may be computationally infeasible. Consequently, as in [20], a reinforcement learning method called proximal policy optimization (PPO) [39] is adopted in DepTaskNet. Note that some existing methods directly use reinforcement learning to make offloading decisions. In contrast, the proposed method uses reinforcement learning to train the encoder-decoder network and then make offloading decisions using the encoder-decoder network.

Algorithm 1 Training Procedure of DepTaskNet

```
Initialize the policy parameters θ_1;
for k = 1 : K do
    // Policy evaluation
    for w = 1 : W do
        λ^w ← Randomly sample a preference;
        G^{wb} ← Randomly sample a batch of instances with a batch size of B, where b = 1, ..., B;
        a^{wb}_{p_j} ← Generate the offloading decision for each task p_j in G^{wb} based on p_{θ_k(λ^w)}(·|G^{wb}), where b = 1, ..., B, j = 1, ..., n;
        Â^{wb}_{p_j} ← Calculate the advantage estimate for each a^{wb}_{p_j}, where b = 1, ..., B, j = 1, ..., n;
        v̂^{wb}_{p_j} ← Obtain the target state value, where b = 1, ..., B, j = 1, ..., n;
    end for
    // Policy improvement
    θ_{k+1} ← Maximize L^{PPO} with H epochs;
end for
Output the optimal policy parameters θ* = θ_{k+1}
```

When determining the decision offloading for task t_{p_j} in G, the state, action, and reward are defined as follows

- The state is defined as $s_{p_j} = \{G, a_{p_1}, \ldots, a_{p_{j-1}}\}$.
- The action is defined as $a_{p_j} \in \{0, 1\}$.
- The reward is defined as [20]

$$r_{p_j} = \lambda_e \frac{\bar{E}^l - \Delta E}{E^l} + \lambda_t \frac{\bar{FT}^l - \Delta FT}{FT^l} \tag{26}$$

where $\bar{E}^l = \frac{E^l}{n}$, $\bar{FT}^l = \frac{FT^l}{n}$, $\Delta E = (1 - a_{p_j})E^l_{p_j} + a_{p_j}E^e_{p_j}$, and $\Delta FT = \max_{i=1,\ldots,j}\{(1 - a_{p_i})FT^l_{p_i} + a_{p_i}FT^d_{p_i}\} - \max_{i=1,\ldots,j-1}\{(1 - a_{p_i})FT^l_{p_i} + a_{p_i}FT^d_{p_i}\}$. Note that $\sum_{j=1}^{n} r_{p_j} = J(\boldsymbol{a}|\lambda)$.

The training procedure of DepTaskNet is given in Algorithm 1. First, the policy parameters (i.e., the encoder-decoder model parameters) are randomly initialized as $\boldsymbol{\theta}_1$. Subsequently, each iteration consists of two main stages: policy evaluation and policy improvement. For policy evaluation, at each iteration k, we randomly sample W preferences (denoted as $\lambda^w, w = 1, \ldots, W$) and B instances (denoted as $G^{wb}, b = 1, \ldots, B$) for each preference λ^w. Then, for each G^{wb}, we use the random sampling method to obtain the corresponding offloading decision $(a^{wb}_{p_j}, j = 1, \ldots, n)$ based on $p_{\boldsymbol{\theta}_k(\lambda^w)}(\cdot|G^{wb})$, and calculate the advantage estimate for each $a^{wb}_{p_j}$:

$$\hat{A}^{wb}_{p_j} = \sum_{k=0}^{n-j+1} (\gamma\varphi)^k \delta_{p_{j+k}}, \quad j = 1, \ldots, n. \tag{27}$$

Here, δ_{p_j} represents the temporal-difference error, which is calculated as

$$\delta_{p_j} = r_{p_j} + \gamma v_{\boldsymbol{\theta}_k(\lambda^w)}(s_{p_{j+1}}) - v_{\boldsymbol{\theta}_k(\lambda^w)}(s_{p_j}). \tag{28}$$

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

TABLE III
PARAMETER SETTINGS OF THE STUDIED MEC SYSTEM.

Variable	Value	Variable	Value
C_i	$[10^7, 10^8]$ cycles	D_i^u, D_i^d	$[5, 50]$ KB
f^l	1 GHz	f^e	10 GHz
c	$1.25 * 10^{-26}$	P^u	1.258 W
β	3	P^d	1.181 W
c_1	0.5	c_2	0.01
γ	0.99	φ	0.95
K	500	ϵ	0.2
H	4	B	256
μ	6		

$v_{\boldsymbol{\theta}_k(\lambda^w)}(s_{p_j})$ is the predicted state value using a value function that shares parameters (i.e., $\boldsymbol{\theta}_k(\lambda^w)$) with the policy. Additionally, the target state value is obtained as

$$\hat{v}_{p_j}^{wb} = \sum_{k=0}^{n-j+1} \gamma^k r_{p_{j+k}}. \tag{29}$$

After that, for policy improvement, we maximize the following objective L^{PPO} to obtain $\boldsymbol{\theta}_{k+1}$ with H epochs via stochastic gradient ascent with Adam [40]. L^{PPO} is defined as

$$L^{PPO}(\boldsymbol{\theta}) = \mathbb{E}_j \left\{ L^{CLIP}(\boldsymbol{\theta}) - c_1 L^{VF}(\boldsymbol{\theta}) + c_2 L^S[p_{\boldsymbol{\theta}(\lambda^w)}](s_{p_j}) \right\} \tag{30}$$

where L^{CLIP} is a clipped target function, L^{VF} is a squared-error loss, $L^S[p_{\boldsymbol{\theta}(\lambda^w)}](s_{p_j})$ is an entropy bonus, and c_1 and c_2 are two constants. Specifically, $L^{CLIP}(\boldsymbol{\theta})$ is defined as

$$\begin{aligned} &L^{CLIP}(\boldsymbol{\theta}) \\ &= \min\left(\frac{p_{\boldsymbol{\theta}(\lambda^w)}}{p_{\boldsymbol{\theta}_k(\lambda^w)}} \hat{A}_{p_j}^{wb}, \text{clip}\left(\frac{p_{\boldsymbol{\theta}(\lambda^w)}}{p_{\boldsymbol{\theta}_k(\lambda^w)}}, 1-\epsilon, 1+\epsilon \right) \hat{A}_{p_j}^{wb} \right). \end{aligned} \tag{31}$$

where $\text{clip}\left(\frac{p_{\boldsymbol{\theta}(\lambda^w)}}{p_{\boldsymbol{\theta}_k(\lambda^w)}}, 1-\epsilon, 1+\epsilon \right)$ restricts that the value of $\frac{p_{\boldsymbol{\theta}(\lambda^w)}}{p_{\boldsymbol{\theta}_k(\lambda^w)}}$ does not exceed the interval $[1-\epsilon, 1+\epsilon]$ and ϵ is a control parameter. In addition, $L^{VF}(\boldsymbol{\theta})$ is defined as

$$L^{VF}(\boldsymbol{\theta}) = \left(v_{\boldsymbol{\theta}(\lambda^w)}(s_{p_j}) - \hat{v}_{p_j}^{wb} \right)^2. \tag{32}$$

The above process is repeated until the maximum number of iterations K is reached, and then the optimal policy parameters $\boldsymbol{\theta}^*$ are output. Hereafter, for the given G and λ, the corresponding offloading decision $\boldsymbol{a}$ can be obtained based on $p_{\boldsymbol{\theta}^*(\lambda)}(\boldsymbol{a}|G)$ by using a greedy strategy.

V. EXPERIMENTAL STUDIES

In this section, we first introduce the parameter settings. Then, the compared methods are briefly presented. Finally, the effectiveness of DepTaskNet is studied and analyzed.

A. Parameter Settings

Similar to [20], this paper used a synthetic DAG generator to generate heterogeneous DAGs. The properties of these DAGs were controlled by the following parameters

- A parameter was used to control the width and height of a DAG;
- A parameter was used to control the number of edges between two levels of a DAG;
- A parameter was used to control the communication-to-computation ratio, which is the ratio between the communication cost and the computation cost.

The above three parameters were randomly selected from $\{0.3, 0.4, 0.5, 0.6, 0.7, 0.8\}$, $\{0.3, 0.4, 0.5,\ 0.6, 0.7, 0.8\}$, and $\{0.3, 0.4, 0.5\}$, respectively.

For each task in the applications, C_i ($i \in \mathcal{T}$) was randomly distributed within $[10^7, 10^8]$ cycles, and both D_i^u and D_i^d were randomly distributed within $[5, 50]$ KB. In the training process of DepTaskNet, we set the number of iterations (denoted as K) to 500. For the reinforcement learning part, the optimized hyperparameters obtained from [41] was applied to our method, and further refined these hyperparameters through random search. Moreover, we set a discount factor γ to 0.99, a GAE parameter φ to 0.95, a value coefficient c_1 to 0.5, an entropy coefficient c_2 to 0.01, and a clip parameter ϵ to 0.2.

The detailed parameter settings of the studied MEC system and DepTaskNet are summarized in Table III [20]. All experiments were carried out with a single RTX 3080 GPU.

B. Compared Algorithms

The following three algorithms were adopted as baselines.

- MOEA/D [42] is a decomposition-based multiobjective evolutionary algorithm. In MOEA/D, a set of uniformly distributed preferences is generated, and a population is utilized to search for a set of optimal offloading decisions, where each offloading decision corresponds to a specific preference.
- DDQNTO [43] is a deep reinforcement learning method, which uses LSTM network, dueling deep Q-network, and double deep Q-network techniques for task offloading.
- Like DepTaskNet, DRLTO [20] employs an encoder-decoder model to approximate the offloading decision policy and applies a policy gradient method to train the encoder-decoder model.

In MOEA/D, 101 preferences were generated, where each preference is represented as $\lambda_i = \frac{i}{100}, i = 0, \ldots, 100$. This approach allows for the simultaneous generation of 101 offloading decisions. Both DDQNTO and DRLTO require one *a priori* preference to transform multiple objectives into a scalar objective. This paper assumed that the models used in DDQNTO and DRLTO were trained with $\lambda = 0.5$. The parameter settings of the three baselines are consistent with the original papers [20], [42], [43].[1]

C. Results and Discussions

1) Comparison on Different Task Numbers: In this section, we first considered nine different task number settings (i.e., n = 10, 15, 20, 25, 30, 35, 40, 45, and 50) and set both R^u and R^d as 6 Mbps. For each task number setting, our

[1]The source code for MOEA/D can be accessed at https://github.com/anyoptimization/pymoo, while the source code for DDQNTO and DRLTO can be found at https://github.com/linkpark/RLTaskOffloading.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

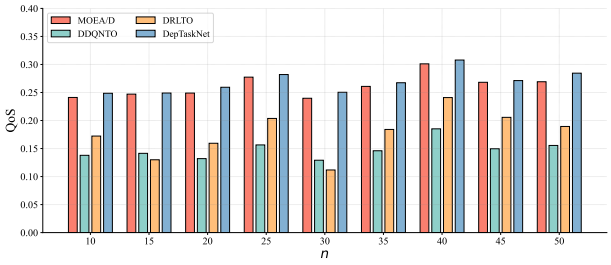

Fig. 7. QoS versus the number of tasks under $R^u, R^d = 6$ Mbps.

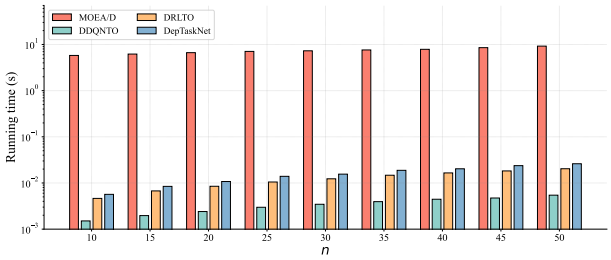

Fig. 8. Running time versus the number of tasks under $R^u, R^d = 6$ Mbps.

study involves 5,000 randomly generated training instances for training the models in DDQNTO, DRLTO, and DepTaskNet. Additionally, we generated 100 test instances and 101 uniformly distributed preferences (i.e., $\lambda_i = \frac{i}{100}, i = 0, \ldots, 100$) to test the performance of the trained models. Due to the fact that the trained model in DepTaskNet is able to accommodate all potential preferences, the offloading decision can be obtained for each preference on each test instance. However, the models in DDQNTO and DRLTO are trained for a specific preference, leading to poor performance and QoS when the preference changes because the offloading decision is the same for different preferences on each instance. MOEA/D directly optimizes the offloading decisions for all 101 preferences on each test instance without the training process. Subsequently, we computed the average QoS of the offloading decisions for different preferences on 100 test instances: $\frac{1}{101*100}\sum_{i=0}^{100}\sum_{k=1}^{100} J(\boldsymbol{a}_k|\lambda_i)$, where $\boldsymbol{a}_k$ is the offloading decision obtained on the kth test instance.

As presented Fig. 7, it is evident that DepTaskNet and MOEA/D outperform DDQNTO and DRLTO across different task number settings since DDQNTO and DRLTO can only provide one offloading decision for different preferences. In most instances, DepTaskNet exhibits slightly better performance than MOEA/D. Additionally, we present the running time of DepTaskNet and three baselines. It is worth noting that DDQNTO, DRLTO, and DepTaskNet only need to perform inference on the trained model to obtain the optimal offloading decisions directly. Therefore, the running time of these three algorithms only includes the inference time. In addition, these models can also continuously train offline to achieve better performance. The running time of MOEA/D represents the duration required by MOEA/D to perform a single run. As depicted in Fig. 8, the running time of DepTaskNet, DDQNTO, and DRLTO is less than 1% compared to MOEA/D. This can be attributed to the fact that these three learning-based

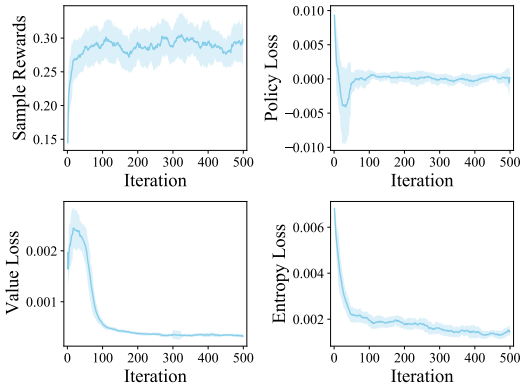

Fig. 9. Convergence of DepTaskNet with a 95% confidence interval under $n = 50$ and $R^u, R^d = 6$ Mbps.

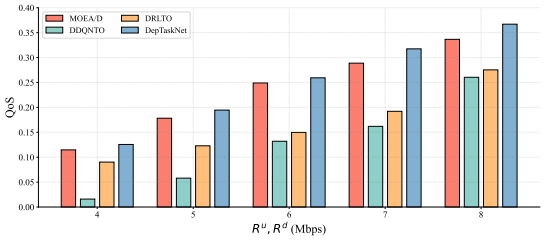

Fig. 10. QoS versus the transmission rate under $n = 20$.

algorithms, i.e., DepTaskNet, DDQNTO, and DRLTO, do not require multiple iterations like MOEA/D and can directly perform inference to make the offloading decision.

Fig. 9 provides the training process of DepTaskNet from different perspectives. The sample reward illustrates the policy evaluation stage during each iteration, in which the policy network samples diverse offloading decisions for different preferences and randomly sampled instances. The policy loss, the value loss, and the entropy loss are used to update the policy network, update the value function network, and balance the exploration and exploitation of the policy, respectively. Considering the case of $n = 50$ and $R^u, R^d = 6$ Mbps, Fig. 9 reveals that the reward experiences rapid growth and stabilizes after approximately 100 iterations, ultimately converging around 400 epochs.

The results mentioned above are summarized as follows. The inability of DDQNTO and DRLTO to accommodate various preferences leads to poor QoS. Therefore, their models usually need to be retrained after the preference changes. On the other hand, MOEA/D directly optimizes the offloading decisions for each test instance under each preference, resulting in decent QoS. However, MOEA/D requires multiple iterations, leading to poor real-time performance. In contrast, DepTaskNet can provide different offloading decisions for various preferences and achieve satisfactory real-time performance.

2) Comparison on Different Transmission Rates: Furthermore, DepTaskNet is compared with the three baselines on different transmission rates: R^u, R^d = 4, 5, 6, 7, and 8 Mbps. Herein, we set the number of tasks as $n = 20$. As shown in Fig. 10, at different transmission rates, DDQNTO still performs the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

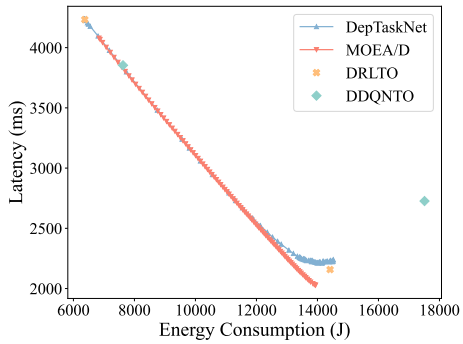

Fig. 11. Energy consumption and latency corresponding to the offloading decisions in different preferences under $n = 50$ and $R^u, R^d = 6$ Mbps.

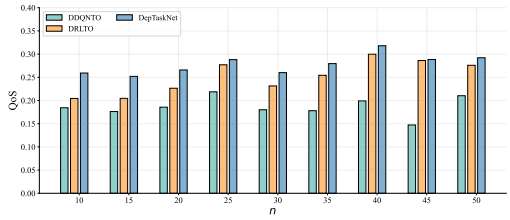

Fig. 12. QoS versus the number of tasks under $R^u, R^d = 6$ Mbps.

worst, followed by DRLTO. However, the performances of MOEA/D and DepTaskNet are significantly better than that of DDQNTO and DRLTO. This further demonstrates the necessity of adjusting offloading decisions based on preferences. In addition, we can observe that DepTaskNet outperforms MOEA/D at different transmission rates, and its advantage increases as the transmission rate increases. These results demonstrate the effectiveness of DepTaskNet on different transmission rates.

3) Energy Consumption and Latency Under Different Preferences: This section examines the energy consumption and latency resulting from offloading decisions under various preferences. As both MOEA/D and DepTaskNet can accommodate different preferences, we compared their performances under 101 uniformly distributed preferences (i.e., $\lambda_i = \frac{i}{100}, i = 0, \ldots, 100$). Moreover, to evaluate the performance of DepTaskNet on each individual objective, we compared DepTaskNet with DDQNTO and DRLTO under two extreme preferences: $\lambda_1 = 0$ and $\lambda_2 = 1$.

From Fig. 11, two significant insights can be drawn.

- A majority of the offloading decisions obtained from MOEA/D and DepTaskNet exhibit similar performance. A small number of offloading decisions obtained from MOEA/D demonstrate lower energy consumption and latency than those obtained from DepTaskNet. However, DepTaskNet offers a wider variety of offloading decisions than MOEA/D. It is worth noting that when the preference is high, the energy consumption of DepTaskNet increases significantly. This is because, as shown in Fig. 11, the offloading decisions perform similarly when the preference is high. This poses a challenge for DepTaskNet in distinguishing these similar offloading decisions with only a single model. In contrast, since MOEA/D makes the offloading decision for each preference separately, it can easily distinguish these offloading decisions. However, in scenarios where the offloading decisions are not similar, the performance of DepTaskNet can surpass that of MOEA/D. Moreover, DepTaskNet does not require many iterations and thus has good real-time performance.
- Compared with DRLTO, DepTaskNet performs slightly worse under $\lambda_1 = 0$ but similarly under $\lambda_2 = 1$. In addition, DepTaskNet performs better than DDQNTO under the two extreme preferences. Since both DDQNTO and DRLTO can only accommodate a preference, they need to be trained on the two preferences separately. In contrast, DepTaskNet does not need to retrain the model after the preference changes. In conclusion, DepTaskNet can efficiently obtain promising results for each individual objective.

4) Comparison on an MEC System with Two Servers: This section extends the studied system to an MEC system with two servers. In this system, each task can be executed not only locally but also remotely on either of the two servers. We assume that the computation capacity of each server is the same. To adapt to this system, we modified the decoder in DepTaskNet, enabling it to make offloading decisions of 0, 1, and 2, corresponding to local execution, remote execution on the first server, and remote execution on the second server, respectively. Subsequently, we compared DepTaskNet with DRLTO and DDQNTO on different task number settings (i.e., n = 10, 15, 20, 25, 30, 35, 40, 45, and 50). As shown in Fig. 12, in the MEC system with two servers, the performance of DepTaskNet is still better than that of the two compared algorithms.

5) A Unified Model to Handle Different Task Number Settings: In Section V-C1, we trained a model for each task number setting, ensuring that the task number remained consistent in both the training and testing instances. Since our model adopts an autoregressive method [44], the model can handle instances of various scales. To this end, we attempted to train a unified model to handle different task number settings simultaneously instead of training different models for different task number settings. As a result, we tried to train a unified model by changing the training procedure without changing the system model and network structure. Specifically, we combined all 45,000 training instances from Section V-C1 to train the unified model. After obtaining the trained unified model (called DepTaskNet-Mix), we employed 100 testing instances to evaluate its performance in each task number setting. As shown in Fig. 13, compared to DepTaskNet, DepTaskNet-Mix yields comparable performance in all task number settings with respect to QoS performance. The results indicate that DepTaskNet-Mix can provide a promising unified model, thereby avoiding the model retraining for different task number settings.

6) Impact of Different Aggregation Functions: This paper employs the weighted sum method to integrate multiple objectives into a scalar objective in (18). Besides the weighted sum method, the Chebyshev method is another scalarization method in multiobjective optimization. To investigate the im-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

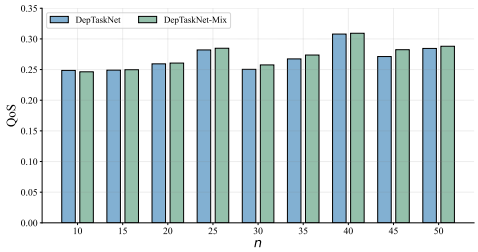

Fig. 13. QoS versus the number of tasks under $R^u, R^d = 6$ Mbps.

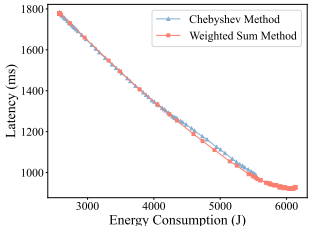

Fig. 14. Energy consumption and latency under different aggregation functions.

pact of scalarization methods on DepTaskNet, we compared the performance of the weighted sum method and the Chebyshev method. As shown in Fig. 14, there is only a slight difference in the performance of the two methods, indicating that the performance of DepTaskNet is not sensitive to the choice of the scalarization method.

VI. Conclusion

This paper investigated multiobjective task offloading that considers task dependencies in MEC systems. The main goal was to simultaneously minimize energy consumption and latency by optimizing the offloading decision without relying on the *a priori* preference. To achieve it, we proposed a neural combinatorial optimization method, which incorporates an encoder to capture relevant information among tasks and a preference-conditioned decoder to determine the offloading decision for each potential preference. Moreover, a reinforcement learning method was employed to optimize the encoder-decoder model. Since the method can directly infer the offloading decision for each preference without the need for model retraining after the preference changes, it offers high flexibility and real-time performance. The effectiveness of the proposed method was validated by comparing it with an evolutionary algorithm and two learning-based algorithms on instances of different scales. In the future, we will try to take multiple UEs into account. Specifically, we can merge multiple DAGs from the UEs into a large DAG by adding a virtual entry point and a virtual exit point and then use the proposed method to make offloading decisions for all tasks.

References

[1] A. N. Saleem, N. M. Noori, and F. Ozdamli, "Gamification applications in e-learning: A literature review," *Technology, Knowledge and Learning*, vol. 27, no. 1, pp. 139–159, 2022.

[2] S. Sonawani, K. Patil, and P. Natarajan, "Biomedical signal processing for health monitoring applications: a review," *International Journal of Applied Systemic Studies*, vol. 10, no. 1, pp. 44–69, 2023.

[3] X. Jiang, F. R. Yu, T. Song, and V. C. Leung, "A survey on multi-access edge computing applied to video streaming: Some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 23, no. 2, pp. 871–903, 2021.

[4] T. Ouyang, R. Li, X. Chen, Z. Zhou, and X. Tang, "Adaptive user-managed service placement for mobile edge computing: An online learning approach," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 1468–1476.

[5] Y. Mao, C. You, J. Zhang, K. Huang, and K. B. Letaief, "A survey on mobile edge computing: The communication perspective," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2322–2358, 2017.

[6] P.-Q. Huang, Y. Wang, and K. Wang, "A divide-and-conquer bilevel optimization algorithm for jointly pricing computing resources and energy in wireless powered mec," *IEEE Transactions on Cybernetics*, vol. 52, no. 11, pp. 12 099–12 111, 2021.

[7] Y. Wang, C.-R. Chen, P.-Q. Huang, and K. Wang, "A new differential evolution algorithm for joint mining decision and resource allocation in a mec-enabled wireless blockchain network," *Computers & Industrial Engineering*, vol. 155, p. 107186, 2021.

[8] Y. Mao, J. Zhang, and K. B. Letaief, "Dynamic computation offloading for mobile-edge computing with energy harvesting devices," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 12, pp. 3590–3605, 2016.

[9] F. Sun, F. Hou, N. Cheng, M. Wang, H. Zhou, L. Gui, and X. Shen, "Cooperative task scheduling for computation offloading in vehicular cloud," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 11, pp. 11 049–11 061, 2018.

[10] Q. Luan, H. Cui, L. Zhang, and Z. Lv, "A hierarchical hybrid subtask scheduling algorithm in uav-assisted mec emergency network," *IEEE Internet of Things Journal*, vol. 9, no. 14, pp. 12 737–12 753, 2021.

[11] P.-Q. Huang, Y. Wang, K. Wang, and Z.-Z. Liu, "A bilevel optimization approach for joint offloading decision and resource allocation in cooperative mobile edge computing," *IEEE Transactions on Cybernetics*, vol. 50, no. 10, pp. 4228–4241, 2020.

[12] L. Huang, S. Bi, and Y.-J. A. Zhang, "Deep reinforcement learning for online computation offloading in wireless powered mobile-edge computing networks," *IEEE Transactions on Mobile Computing*, vol. 19, no. 11, pp. 2581–2593, 2020.

[13] A. Gao, S. Zhang, Y. Hu, W. Liang, and S. X. Ng, "Game-combined multi-agent drl for tasks offloading in wireless powered mec networks," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 7, pp. 9131–9144, 2023.

[14] P. Dai, K. Hu, X. Wu, H. Xing, and Z. Yu, "Asynchronous deep reinforcement learning for data-driven task offloading in mec-empowered vehicular networks," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[15] J. Tan, R. Khalili, H. Karl, and A. Hecker, "Multi-agent distributed reinforcement learning for making decentralized offloading decisions," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 2098–2107.

[16] F. Chai, Q. Zhang, H. Yao, X. Xin, R. Gao, and M. Guizani, "Joint multi-task offloading and resource allocation for mobile edge computing systems in satellite iot," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 6, pp. 7783–7795, 2023.

[17] N. Zhao, Z. Ye, Y. Pei, Y.-C. Liang, and D. Niyato, "Multi-agent deep reinforcement learning for task offloading in uav-assisted mobile edge computing," *IEEE Transactions on Wireless Communications*, vol. 21, no. 9, pp. 6949–6960, 2022.

[18] J. Wang, C. Jiang, H. Zhang, Y. Ren, K.-C. Chen, and L. Hanzo, "Thirty years of machine learning: The road to pareto-optimal wireless networks," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1472–1514, 2020.

[19] J. Yan, S. Bi, and Y. J. A. Zhang, "Offloading and resource allocation with general task graph in mobile edge computing: A deep reinforcement learning approach," *IEEE Transactions on Wireless Communications*, vol. 19, no. 8, pp. 5404–5419, 2020.

[20] 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 Transactions on Computers*, vol. 71, no. 10, pp. 2449–2461, 2022.

[21] X. Zhang, W. Wu, Z. Zhao, J. Wang, and S. Liu, "Rmddqn-learning: Computation offloading algorithm based on dynamic adaptive multi -

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication.
Citation information: DOI: 10.1109/TVT.2025.3546914, IEEE Transactions on Vehicular Technology

objective reinforcement learning in internet of vehicles," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 9, pp. 11 374–11 388, 2023.
[22] M.-H. Chen, B. Liang, and M. Dong, "A semidefinite relaxation approach to mobile cloud offloading with computing access point," in *2015 IEEE 16th International Workshop on Signal Processing Advances in Wireless Communications (SPAWC)*. IEEE, 2015, pp. 186–190.
[23] M.-H. Chen, M. Dong, and B. Liang, "Joint offloading decision and resource allocation for mobile cloud with computing access point," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 3516–3520.
[24] X. Guo, R. Singh, T. Zhao, and Z. Niu, "An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems," in *2016 IEEE International Conference on Communications (ICC)*. IEEE, 2016, pp. 1–7.
[25] V. Di Valerio and F. L. Presti, "Optimal virtual machines allocation in mobile femto-cloud computing: An mdp approach," in *2014 IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*. IEEE, 2014, pp. 7–11.
[26] A. Bozorgchenani, F. Mashhadi, D. Tarchi, and S. A. Salinas Monroy, "Multi-objective computation sharing in energy and delay constrained mobile edge computing environments," *IEEE Transactions on Mobile Computing*, vol. 20, no. 10, pp. 2992–3005, 2021.
[27] Y. Li, X. Zhu, N. Li, L. Wang, Y. Chen, F. Yang, and L. Zhai, "Collaborative content caching and task offloading in multi-access edge computing," *IEEE Transactions on Vehicular Technology*, vol. 72, no. 4, pp. 5367–5372, 2023.
[28] Z. Xiao, J. Shu, H. Jiang, J. C. S. Lui, G. Min, J. Liu, and S. Dustdar, "Multi-objective parallel task offloading and content caching in D2D-aided MEC networks," *IEEE Transactions on Mobile Computing*, vol. 22, no. 11, pp. 6599–6615, 2023.
[29] S. Ma, S. Song, L. Yang, J. Zhao, F. Yang, and L. Zhai, "Dependent tasks offloading based on particle swarm optimization algorithm in multi-access edge computing," *Applied Soft Computing*, vol. 112, p. 107790, 2021.
[30] X. Li, M. Abdallah, S. Suryavansh, M. Chiang, K. T. Kim, and S. Bagchi, "Dag-based task orchestration for edge computing," in *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2022, pp. 23–34.
[31] R. T. Marler and J. S. Arora, "The weighted sum method for multi-objective optimization: new insights," *Structural and Multidisciplinary optimization*, vol. 41, pp. 853–862, 2010.
[32] H. Cao and J. Cai, "Distributed multiuser computation offloading for cloudlet-based mobile cloud computing: A game-theoretic machine learning approach," *IEEE Transactions on Vehicular Technology*, vol. 67, no. 1, pp. 752–764, 2017.
[33] S. Liu, Y. Zhang, K. Tang, and X. Yao, "How good is neural combinatorial optimization? a systematic evaluation on the traveling salesman problem," *IEEE Computational Intelligence Magazine*, vol. 18, no. 3, pp. 14–28, 2023.
[34] W. Song, X. Chen, Q. Li, and Z. Cao, "Flexible job-shop scheduling via graph neural network and deep reinforcement learning," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1600–1610, 2023.
[35] X. Lin, Z. Yang, and Q. Zhang, "Pareto set learning for neural multi-objective combinatorial optimization," in *International Conference on Learning Representations*, 2022.
[36] X. Lin, Z. Yang, X. Zhang, and Q. Zhang, "Pareto set learning for expensive multi-objective optimization," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 19 231–19 247.
[37] A. Graves and J. Schmidhuber, "Framewise phoneme classification with bidirectional lstm and other neural network architectures," *Neural networks*, vol. 18, no. 5-6, pp. 602–610, 2005.
[38] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
[39] H. An and L. Wang, "Robust topology generation of internet of things based on ppo algorithm using discrete action space," *IEEE Transactions on Industrial Informatics*, vol. 20, no. 4, pp. 5406–5414, 2024.
[40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.
[41] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
[42] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition," *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, 2007.
[43] M. Tang and V. W. Wong, "Deep reinforcement learning for task offloading in mobile edge computing systems," *IEEE Transactions on Mobile Computing*, vol. 21, no. 6, pp. 1985–1997, 2022.
[44] A. Graves, "Generating sequences with recurrent neural networks," *arXiv preprint arXiv:1308.0850*, 2013.

Xiang-Jie Xiao received the B.Eng. degree in Electronic Science and Technology from Central South University of Forestry and Technology, Changsha, China, in 2021, and the M.Eng. degree in Automation from Central South University, Changsha, China, in 2024. He is currently pursuing the Ph.D. degree in Computer Science at Singapore Management University, Singapore. His research interests include scheduling and neural combinatorial optimization.

Yong Wang (Senior Member, IEEE) received the Ph.D. degree in control science and engineering from the Central South University, Changsha, China, in 2011.

He is a Professor with the School of Automation, Central South University, Changsha, China. His current research interests include intelligent learning and optimization and their interdisciplinary applications.

Dr. Wang is an Associate Editor of the *IEEE Transactions on Evolutionary Computation* and the *Swarm and Evolutionary Computation*. He was a recipient of Cheung Kong Young Scholar by the Ministry of Education, China, in 2018, and a Web of Science highly cited researcher in Computer Science in 2017 and 2018.

Pei-Qiu Huang (Member, IEEE) received the B.S. degree in automation and the M.S. degree in control theory and control engineering both from the Northeastern University, Shenyang, China, in 2014 and 2017, respectively, and the Ph.D. degree in control science and engineering, Central South University, Changsha, China, in 2021.

He is an Associate Professor with the School of Automation, Central South University, Changsha, China. His current research interests include evolutionary computation, bilevel optimization, and their applications.

Kezhi Wang (Senior Member, IEEE) received the Ph.D. degree from the University of Warwick, U.K. He is currently a Professor with the Department of Computer Science, Brunel University London, U.K. His research interests include wireless communications, mobile edge computing, and machine learning. He is a Royal Society Industry Fellow and a Clarivate Highly Cited Researcher in 2023.