End-Edge-Cloud Heterogeneous Resources Scheduling Method Based on RNN and Particle Swarm Optimization

Haijie Wu, Wangbo Shen, Weiwei Lin, *Senior Member, IEEE*, Wei Li, *Senior Member, IEEE*, and Keqin Li, *Fellow, IEEE*

***Abstract*—Task scheduling in cloud computing is a challenging but crucial task for ensuring service quality and load balance. Mainstream scheduling algorithms, such as heuristic algorithms and reinforcement learning, have made progress in this area. However, online task scheduling algorithms, such as reinforcement learning, can pose computational challenges in scenarios with limited computational power and heterogeneous resources. Heuristic algorithms, which are more suitable for offline scheduling where the types and quantities of tasks are known in advance, also require substantial computational resources for online scheduling. In this work, we propose the end-edge-cloud (EEC) heterogeneous resources scheduling method (EHRSM) based on a recurrent neural network (RNN) model and particle swarm optimization (PSO). EHRSM uses an RNN model trained on a dataset generated by dynamic programming to recognize and cache online tasks, efficiently transforming online task scheduling into offline scheduling. Additionally, a PSO algorithm with Cantor expansion (CE) for coding optimization is used to complete the offline scheduling. Experimental results show that the method is effective in converting online scheduling to offline scheduling, reducing the average task completion time and waiting time. Compared with existing online scheduling methods, EHRSM reduces task completion time by up to 48.24%.**

***Index Terms*—Cantor expansion, end-edge-cloud, particle swarm optimization, recurrent neural network.**

I. Introduction

With the explosive growth of the number of terminal devices and network service demand soaring, emerging industries such as driverless and smart home have a higher and higher demand for computing services. The traditional framework of cloud computing is unable to provide the corresponding efficient computing services. Therefore, many emerging cloud computing frameworks such as EEC have appeared. EEC aims to shorten the distance between users and fully cooperate with the computing resources of each device, to bring low latency services.

In recent years, there have been many scientific researches and applications related to EEC. Yang et al. presented an EEC framework that optimizes the efficiency of data processing and deployment of nodes, which tackles the difficulty of the calculation and transmission of data produced by healthcare devices [1]. Liu et al. proposed an adaptive DNN inference acceleration framework, accelerating DNN inference by fully utilizing the EEC collaborative computing, and reducing end-to-end inference latency caused by long wide-area massive data transmission and performance degeneration [2]. Ding et al. proposed an EEC collaborative emotion perception network model, and its cloud control terminal performs online training to dynamically adjust the parameters of the model in edge devices which efficiently perceives the emotion of the driver and reduces the occurrence of traffic accidents [3]. Duan et al. presented an exhaustive survey on the distributed artificial intelligence facilitated by EEC computing and showed the benefits of the EEC in supporting distributed AI [4]. These studies show that EEC is used in many scenarios, and many industries with lower network latency requirements and higher service quality requirements combine many artificial intelligence technologies with EEC to achieve better results. It also suggests that the EEC is becoming a research hotspot, optimized by more and more technologies so that it can be better applied in various scenarios.

Like traditional cloud computing frameworks, EEC also has resource allocation and scheduling issues. In traditional cloud computing services, users deliver tasks to the cloud, and the framework automatically completes the allocation of computing resources without user involvement. Resource allocation and scheduling are extremely important in this process, which determines how and how much computing resources the tasks will be executed and affects the task completion time, resource utilization, and device load. Therefore, the scheduling problem has become a valuable and difficult topic to solve. There is several scientific work related to scheduling tasks. In [5], a swarm-intelligence-based approach, specifically a hybridized

Received 17 March 2024; revised 27 September 2024; accepted 23 November 2024. Date of publication 27 November 2024; date of current version 22 April 2025. This work is supported by National Natural Science Foundation of China (62072187), Guangzhou Development Zone Science and Technology Project (2023GH02) and the Major Key Project of PCL, China under Grant PCL2023A09. The associate editor coordinating the review of this article and approving it for publication was N. Kamiyama. *(Corresponding author: Weiwei Lin.)*

Haijie Wu and Wangbo Shen are with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China (e-mail: 202030442496@mail.scut.edu.cn; 202010107337@mail.scut.edu.cn).

Weiwei Lin is with the School of Computer Science and Engineering, South China University of Technology, Guangzhou 510006, China, and also with the Department of New Networks, Pengcheng Laboratory, Shenzhen 518055, China (e-mail: linww@scut.edu.cn).

Wei Li is with the School of Computer Science, The University of Sydney, Sydney, NSW 2006, Australia (e-mail: weiwilson.li@sydney.edu.au).

Keqin Li is with the Department of Computer Science, State University of New York at New Paltz, New Paltz, NY 12561 USA (e-mail: lik@newpaltz.edu).

Digital Object Identifier 10.1109/TNSM.2024.3507017

bat algorithm, is proposed as a method for approximating solutions for multi-objective task scheduling problems in the cloud environment. A new task scheduling method, I-FASC, was proposed for the characteristics of tasks and resources, to deal with the less computing resources and weak processing ability of fog devices [6]. The author in [7] proposed a modified PSO technique, which focused on average scheduling length and the ratio of successful execution. These studies reveal the importance of scheduling for improving cloud computing services, impacting the experience of users and device costs. In addition, scheduling policies are often different with different application environments, so it is necessary to design efficient scheduling policies for different scenarios.

The goal of scheduling is often based on two considerations. On the one hand, from the user's perspective, short task execution times will improve the user's experience. This often includes some explicit metrics such as the time between task delivery and completion, the sum of task wait times, etc. [8], [9], [10]. On the other hand from the performance loss of equipment, clusters tend to pursue a load-balanced state, so it's improper to overload powerful servers and leave mediocre ones idle which can lead the server overloaded to shorter life and degrade performance. Therefore, many scheduling algorithms also consider the problem of load balancing [11], [12], [13]. In addition, there are some scheduling algorithms, which are oriented to other optimization goals according to their different application environments. Such as methods based on reducing energy consumption [14], focusing on fault-tolerance [15], facing fair scheduling [16], and so on have been studied.

However, it is precisely because of the emergence of the EEC, that the devices managed by the EEC show diversity in performance, and the computing resources are gradually heterogeneous, which leads to a decrease in system resource utilization and the difficulty of coordinating heterogeneous resources. Under such challenges, traditional algorithms will find it difficult to play a good role in EEC. Therefore, many scheduling algorithms for heterogeneous resources have been proposed in recent years [17], [18], [19]. These achievements take into account the heterogeneity of resources and improve the performance of scheduling to a certain extent.

Despite this, there are still many problems in solving the scheduling of heterogeneous resources in the EEC. In an online scheduling system, tasks are dispatched at irregular intervals. Because of the heterogeneity of resources, there are often higher dimensions in the expression of data, and the calculation of scheduling is more complicated. Some classic algorithms, such as greedy algorithms, have significant room for improvement in scheduling performance due to limited considerations. Reinforcement learning methods, which explore the optimization space of scheduling from a more comprehensive perspective [20], are often applied to online task scheduling. However, reinforcement learning faces problems of lengthy training and a large amount of data demand, and there are also shortcomings in the complex and ever-changing EEC. Many metaheuristics have been applied to cloud computing scheduling, and have shown good performance, but most of them can only be applied to offline task scheduling, and because of the long execution time of the algorithm, they are rarely used in real-time scenarios such as EEC or edge computing.

In this paper, we summarize several problems in the scheduling of heterogeneous resources in the EEC:

- Due to the heterogeneity of resources, modeling tasks and cluster resources is more complex and high-dimensional, and there is currently a lack of effective scheduling algorithms for collaborative EEC heterogeneous resources.
- Compared to traditional cloud computing, the node information of EEC is more prone to change, so scheduling algorithms need to be as lightweight and stable as possible to cope with changing tasks and node structures and reduce algorithm costs, which is difficult for mainstream online scheduling algorithms to achieve.

Given this, we propose an End-edge-cloud Heterogeneous Resources Scheduling Method (EHRSM) based on RNN and Particle Swarm Optimization. This method integrates real-time heterogeneous resources of clusters to facilitate efficient scheduling of online tasks. The effectiveness of this method within the EEC context is substantiated through experimentation. For this method, we summarize the contributions and innovations points as follows:

1) We propose an RNN-based method for batching online tasks, which converts online task scheduling into batch scheduling, and a computationally fast and efficient batch task scheduling method based on PSO, which satisfies the timeliness of EEC scheduling while making efficient scheduling. Our proposed method does not require a large amount of computing resources and long training time, achieving resource collaboration in EEC.
2) In the training of RNN, we use a data labeling method based on dynamic programming to eliminate errors in manually labeling datasets and give RNN the ability to batch tasks correctly. In addition, we optimize the encoding and inertia weight of PSO using CE and logistic function, respectively, to achieve excellent scheduling performance while fast convergence.
3) We evaluate the feasibility and effectiveness of the proposed method through comparative experiments, proving that it can be efficiently applied to heterogeneous resource scheduling in EEC and improve service quality.

The rest of this paper is organized as follows: Section II introduces some background and related work. Section III presents the formulation of the scheduling problems. Section IV provides a detailed description of the design and analysis of the proposed method. Section V carries out experiments and the analysis of the results. Section VI concludes this paper.

II. Background and Related Work

A. EEC Framework and Its Heterogeneity

With the development of the Internet, the explosive growth of terminal equipment has aggravated network congestion and high latency, while emerging industries such as autonomous driving and smart homes have higher and higher demand for network quality. A framework that can cooperate with terminal

equipment, edge servers, and cloud servers, called EEC, has emerged to reduce the transmission time of computing data and improve the quality of service. In this framework, the cloud server usually takes charge of the whole system, and its responsibilities include the management of the system structure, the storage of important data, and complex computational tasks such as model training. As a node closes to the terminal, the edge server can transmit data quickly, so it is often used for real-time data processing. The computing resources of the terminals are also coordinated to obtain data from outside the device and process information received from other terminals. These three parts cooperate and perform their duties to make full use of the resources and performance of devices.

EEC is currently used in many industries such as medical devices, traffic safety, etc. [1], [3] where large amounts of data and complex calculations need to be processed. This is due to the efficient collaboration of nodes on the EEC, including data collaboration, computing resource collaboration, and so on. However, it is precisely this collaboration that requires systems deploying EEC to model the heterogeneous resources of the nodes it manages. For example, many devices carry the GPU or NPU which is different from the CPU computing resources and they usually have a lot of differences in performance. This means that if there are no appropriate scheduling algorithms, it will result in low resource utilization and long waiting times for computing services. In recent years, there has been some research on scheduling algorithms for EEC or similar scenarios. In [21], a Content-Aware Task Scheduling Algorithm framework was proposed to solve the scheduling problem in heterogeneous mobile edge cloud paradigms. Reference [22] proposed an adaptive application-aware task scheduling algorithm for running over heterogeneous edge cloud. The use of ant colony algorithm to optimize the cloud-edge collaborative task scheduling was proposed in [23]. These studies have further promoted the development of EEC.

B. Optimization Methods and Scheduling Applications of PSO

As a swarm intelligence algorithm with low computational complexity and fast convergence, PSO has been widely studied and there is much literature that optimizes it. Inertia weight, one of the parameters in PSO, has a significant impact on the convergence performance of PSO, and much literature has optimized the inertia weight. In [24], a time-vary adaptive inertia weight parameter was proposed to increase the diversity of particles and showed that the proposed method has succeeded in finding a better UAV path. Also, different particles can have different inertia weights according to [25], which proposed an inertia weight adjustment method based on the optimal fitness value of individual particles. Reference [26] proposed 5 strategies for inertia weight reduction, including sigmod decreasing, simulated annealing, etc., and compares their performance advantages and disadvantages.

Another parameter of PSO, the acceleration constant, is also the direction for optimizing PSO. In [27], the acceleration constant is designed to be adaptive and automatically change over time, while another literature [28] designed the acceleration constant based on sine cosine. However, the optimization of acceleration coefficients and inertia weight is not contradictory to each other, so there is some literature that simultaneously optimizes both [29], [30], [31].

PSO is widely used in task scheduling in cloud computing scenarios due to its strong optimization ability and fast convergence. Generally speaking, for a group of tasks and a group of virtual machines, the scheduling situation of the task is encoded, and then PSO is used to obtain the scheduling strategy with different optimization targets [26], [32]. Many applications of PSO are combined with other algorithms or optimization methods [33], [34], [35], [36]. However, as an offline algorithm, PSO can only schedule known task sets and therefore is unsuitable to complete online task scheduling.

III. Problem Formulation

In the cloud computing environment, users submit tasks to the cloud and wait for the tasks to be executed. We assume that tasks are executed in an environment where EEC is deployed and each user request is handed over to the cloud server for scheduling. Task information, including CPU, memory, and execution time, is crucial for scheduling and can be obtained from historical running information. Due to the inclusion of cloud, edge, and terminal devices in the cluster, there is a diversity among the devices. Many devices have inconsistent architectural compositions, such as CPUs with arm64 and x86 architectures. Even CPUs with the same architecture may have different task execution times due to quality differences. Therefore, it is also necessary to obtain execution information for the same task on all nodes. More importantly, the heterogeneity of the EEC determines that there is more than one way to run tasks. Many servers are equipped with chips such as GPUs and NPUs that outperform CPUs. For tasks that require AI model training and inference, using GPUs can lead to better execution efficiency. Therefore, the difference in execution performance of tasks using different chips of the same device also needs to be considered in scheduling.

We first model the problem. In an EEC system, the nodes are denoted as $E = \{e_1, e_2, \ldots, e_{|E|}\}$ and the sequence of tasks is $N = \{n_1, n_2, \ldots, n_{|N|}\}$. For these tasks, their arrival time is denoted as $S = \{s_1, s_2, \ldots, s_{|N|}\}$. There are R types of resources we are considering, including CPU usage, CPU memory remaining, GPU usage, etc. The number of running modes a task can support is K, such as using CPU mode, GPU mode, NPU mode, etc. When a task is executed using GPU mode, the task will occupy most of the GPU chip, and a small portion of the CPU chip, while other chips will not be used. Therefore, different running modes will consume different computational resources. Note that in the actual scheduling process, we cannot predict the arrival time and type of tasks, but the resource requirements and execution time of each task can be predicted from historical records. Specifically, the running resource requirement of a task is denoted as $Q_{i,j,k}^{(r)}$, which represents the rth resource where the ith task runs in the kth mode on the jth node and $n_i \in N, e_j \in E, 1 \le k \le K, 1 \le r \le R$, and $Q_{i,j,k}^{(R+1)}$ represents the corresponding runtime of the task.

If the task cannot run in this mode on the node, then $Q_{i,j,k}^{(r)} = +\infty, 1 \le r \le R$ and $Q_{i,j,k}^{(R+1)} = 0$. The real-time resource of a node is denoted as $A_j^{(r)}(\tau)$, which represents the remaining amount of the rth resource in the jth node at time τ and $e_j \in E, 1 \le r \le R$. For these tasks, the scheduling result is denoted as $G = \{g_1, g_2, \ldots, g_{|N|}\}$. As mentioned earlier, the scheduling result should contain the running node and the running mode. However, due to the limited resources, each arriving task does not always get enough resources to run immediately, and thus the scheduling result also contains the waiting time, i.e., $g_i =< \xi_i, \zeta_i, \nu_i >$, where ξ_i represents the node number where the ith task runs, ζ_i represents the mode the task run, and ν_i represents the time the task needs to wait, and $e_{\xi_i} \in E, 1 \le \zeta_i \le K$. The time from submission to completion of task n_i includes the following parts: the transmission delay $T_i^{\uparrow}$ from the user to the cloud scheduler, the waiting time T_i^w for task scheduling, the transmission delay $T_{\xi_i}^{\downarrow}$ from the cloud scheduler to the node ξ_i, the waiting time ν_i for the task to run on the node, and the execution time $Q_{i,\xi_i,\zeta_i}^{(R+1)}$ of the task. Therefore, the completion time $T_i^{\triangle}$ of task n_i is represented as

$$T_i^{\triangle} = T_i^{\uparrow} + T_i^w + T_{\xi_i}^{\downarrow} + \nu_i + Q_{i,\xi_i,\zeta_i}^{(R+1)} \tag{1}$$

The goal of the scheduling policy is to find the optimal G with the minimum total task completion time (TCT) and total waiting time (TWT), which are computed by the following equations:

$$TCT = \sum_{i=1}^{|N|} T_i^{\triangle} \tag{2}$$

$$TWT = \sum_{i=1}^{|N|} \left(T_i^{\uparrow} + T_i^w + T_{\xi_i}^{\downarrow} + \nu_i \right) \tag{3}$$

Our goal is expressed as

$$\min \ \alpha \times TCT + (1-\alpha) \times TWT$$
$$\text{s.t.} \begin{cases} A_{\xi_i}^{(r)}(s_i + \nu_i) \ge Q_{i,\xi_i,\zeta_i}^{(r)} \\ A_j^{(r)}(\tau) \ge 0, \ \forall r \le R, \forall \tau \ge 0, \forall e_j \in E \end{cases} \tag{4}$$

α is used to weight TCT and TWT and $0 \le \alpha \le 1$. Scheduling policy may make tasks wait a long time to run on nodes with fast execution to reduce completion time, or it may minimize the waiting time but result in an increase in completion time. Therefore TCT and TWT need to be weighted even though they are related to each other. The constraint of resources reflects the limited resources of EEC, and tasks can only be executed on nodes with sufficient resources. Unlike existing work, EEC requires the efficient collaboration of heterogeneous resources, not just CPU and memory. The multiple running modes of tasks determine the utilization performance of heterogeneous resources, which leads to more complex scheduling scenarios.

IV. Algorithm Design and Analysis

Based on the above modeling, we propose EHRSM, a scheduling method for EEC heterogeneous resources, to achieve the goal of Eq. (4). The overall process of EHRSM is shown in Fig. 1. EHRSM uses an RNN model to adaptively batch online tasks and employs PSO for batch scheduling, transforming online scheduling into offline scheduling. Next, we describe in detail how these techniques can be used to achieve efficient scheduling.

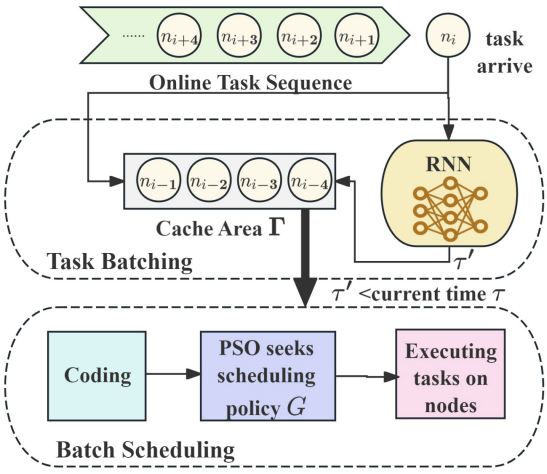

Fig. 1. Process diagram of EHRSM. The online task n_i first calculates the scheduling time τ' by RNN. When τ' is less than the current time τ, all tasks in the cache area Γ are scheduled by PSO. Otherwise, task n_i will be temporarily stored in the cache area.

A. Process of Task Batching

We batch tasks based on their runtime and arrival time. When the scheduling timing is triggered, all cached tasks are scheduled. Generally speaking, for tasks with a long runtime, we hope it can wait for a longer time to accumulate more tasks and make better scheduling strategies. For tasks with short runtimes, we hope they can be executed as soon as possible because making the cache time longer than the task's runtime can bring a poor experience. The batch method for tasks requires providing appropriate scheduling time for each task.

For the above description, we design an RNN model as a task buffer. The RNN model can analyze and calculate current inputs based on historical inputs and is easy to train and converge for its simple construction. Enabling the RNN model to have the ability to adaptively batch tasks is one of the key components of EHRSM. RNNs can only have the ability to batch tasks if they are trained with the right dataset. Therefore, we focus on obtaining the right dataset, including the sequence of tasks and the optimal scheduling time as labels, which is one of the innovations of this paper. The label acquisition of datasets in traditional neural network models is usually based on manual annotations, nevertheless, the optimal scheduling time of a task sequence is difficult to obtain through manual annotations, since we cannot intuitively and artificially batch the overall task sequences to obtain efficient scheduling performance. Therefore, we propose the following method for optimal scheduling time labeling of task sequences, which is based on a dataset with known task arrival times and types.

The process of labeling the dataset is shown in Fig. 2. Assuming the task sequence in the dataset is

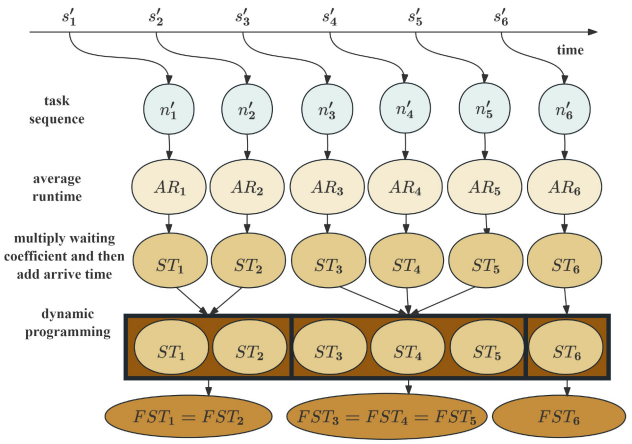

Fig. 2. The generation process of RNN training dataset. The task sequence is first calculated for the average running time, and then the scheduling time of the task is further calculated. To segment the labels of the data, dynamic programming divides the task sequence to minimize the difference in scheduling time for each segment. The final scheduling time is the average of the scheduling times of the tasks in the same segment.

$N' = \{n'_1, n'_2, \ldots, n'_{|N'|}\}$ and the arrival time is $S' = \{s'_1, s'_2, \ldots, s'_{|N'|}\}$. Due to the varying running times of a task at different nodes and running modes, we calculate the average running time of the task. Define $\gamma_{i,j,k}$ as the maximum number of times the *i*th task can run in parallel in the *k*th mode when the *j*th node is empty:

$$\gamma_{i,j,k} = \min_{1 \le r \le R} \left\lfloor \frac{A_j^{(r)}(0)}{Q_{i,j,k}^{(r)}} \right\rfloor \tag{5}$$

where $1 \le i \le |N'|, 1 \le j \le |E|, 1 \le k \le K$. Define $\delta_{i,j,k}$ as the popularity of the *i*th task to run in the *k*th mode on the *j*th node:

$$\delta_{i,j,k} = \frac{\gamma_{i,j,k}}{\sum_{q=1}^{|E|} \sum_{l=1}^{K} \gamma_{i,q,l}} \tag{6}$$

The higher the popularity, the more likely the task is to run in this mode on that node. Therefore, the average runtime of the *i*th task is defined as AR_i:

$$AR_i = \sum_{j=1}^{|E|} \sum_{k=1}^{K} \delta_{i,j,k} \times Q_{i,j,k}^{(R+1)} \tag{7}$$

As mentioned earlier, tasks with longer runtimes are suitable for scheduling later. Therefore, the scheduling time ST_i of the *i*th task is calculated using the following equation:

$$ST_i = s'_i + \beta \times AR_i \tag{8}$$

where β is the waiting coefficient. The larger the β, the later the scheduling time of the task. However, the final labels should be segmented, as they indicate which tasks need to be divided into the same batch. For example, there are a total of 7 tasks in the task sequence, labeled as {1, 1, 1, 3, 3, 4, 4}, which indicates that tasks 1-3 are scheduled in the first second, while tasks 4-5 are scheduled in the third second. Therefore, further, assuming that the scheduling time of all tasks is $\Lambda = \{ST_1, ST_2, \ldots, ST_{|N'|}\}$. We assume to schedule a total of $|M|$ times to complete these tasks, which means that Λ will be divided into $|M|$ segments. We hope that the difference in a segment is as small as possible, as tasks in the same segment will be scheduled at the same time. Since variance can well reflect the degree of difference in a sequence, we use variance to calculate the difference in each segment. The optimal division scheme can minimize the variance of all segments. We define the division scheme as $M = \{m_1, m_2, \ldots, m_{|M|}\}$, where $1 < m_1 < m_2 < \cdots < m_{|M|} = |N'|$. The problem description is as follows:

$$\min \sum_{i=1}^{|M|} \sigma^2_{m_{i-1}+1 \sim m_i}(\Lambda) \tag{9}$$

where $m_0 = 0$, $\sigma^2_{m_{i-1}+1 \sim m_i}(\Lambda)$ denotes the variance of the subarray of Λ subscripts from $m_{i-1}+1$ to m_i. For this problem, we use a dynamic programming approach to solve it. Specifically, define $D_{i,j}$ as the minimum total variance of the first *i* tasks divided into *j* segments, hence the following transfer equation:

$$D_{i,j} = \begin{cases} 0, & i = 0 \&\& j = 0 \\ +\infty, & j > i \\ \min\limits_{j-1 \le k \le i-1} \left(D_{k,j-1} + \sigma^2_{k+1 \sim i}(\Lambda) \right), & otherwise \end{cases} \tag{10}$$

Ultimately, $D_{|N'|,|M|}$ denotes the sum of the minimum variances of dividing Λ into $|M|$ segments, whose corresponding division schemes are obtained during the transfer process. Define the final scheduling time FST_i of the *i*th task as the mean value of the segment in which the task is located:

$$FST_i = \mu_{m_{j-1}+1, m_j}(\Lambda), \ m_{j-1} + 1 \le i \le m_j \tag{11}$$

where $\mu_{m_{j-1}+1, m_j}(\Lambda)$ denotes the mean of the subarray of Λ subscripts from $m_{j-1}+1$ to m_j, $2 \le j \le |M|$. Therefore, the *i*th sample input for training RNN is $< AR_i, s'_i >$ and the output is $< FST_i >$.

The trained RNN will be used for batching the actual online task sequences, as shown in Fig. 1. Γ is defined as the cache area for batch processing tasks. For task n_i that arrives at time s_i, it is placed in the cache area, i.e., $\Gamma \cup \{n_i\}$. We calculate the average running time AR_i based on the task type, input the trained RNN, and update the scheduling timing τ':

$$\tau' = \min\left(\tau', RNN(< AR, s >)\right) \tag{12}$$

where $RNN(< AR, s >)$ represents the output of the RNN model when the input is $< AR, s >$. If τ' is greater than the current time, continue to wait for the task to arrive. Otherwise, schedule all tasks in Γ and clear Γ.

We further describe why we use this approach. For each task, we can compute the scheduling time *ST*, which, although it takes into account the arrival time and the running time of the task, does not take into account the effects of the tasks before and after. Therefore, we further compute the *FST* to synthesize the before and after information of the task sequence to get the optimal scheduling time. Since future task information cannot be accurately obtained in real scheduling scenarios, for the current task, we can only utilize the information before that task, however, RNN can handle the problem well. On the one

hand, RNN can predict future task characteristics, on the other hand, RNN extracts the features of the sequence of arrived tasks to give the proper scheduling time. Traditional caching approaches may be based on either a number threshold or a time threshold, but either one is not very adaptable, as we will demonstrate in Section V.

B. Batch Scheduling

For the batch tasks obtained using RNN batching, we use PSO for globally optimal scheduling. Similar to edge computing, scheduling in EEC scenarios also needs to take into account the execution time of the scheduling algorithm itself, while PSO converges quickly and can find the optimal solution in a shorter time. PSO is based on the following two equations:

$$vel_i^{(k)} = w \cdot vel_i^{(k-1)} + \varphi_1 rand()\left(pbest_i - pos_i^{(k-1)}\right) + \varphi_2 rand()\left(gbest - pos_i^{(k-1)}\right) \tag{13}$$

$$pos_i^{(k)} = pos_i^{(k-1)} + vel_i^{(k)} \tag{14}$$

where $vel_i^{(k)}$ indicates the velocity of the *i*th particle at the *k*th round, *w* indicates the inertia weight, $pos_i^{(k)}$ indicates the position of the *i*th particle at the *k*th round, φ_1 and φ_2 indicate the acceleration coefficients, *rand*() indicates a random number from 0 to 1, $pbest_i$ indicates the optimal historical position of the *i*th particle, and *gbest* indicates the optimal historical position of all particles. Despite its advantage of fast convergence, PSO is used for online task scheduling in the proposed EHRSM, which has greater requirements for execution speed and optimization performance, and naive PSO is difficult to meet these requirements. Therefore, we improve the coding and inertia weights of PSO to provide better scheduling performance.

For a scheduling strategy, its content generally includes the mapping of tasks to nodes. However, due to the heterogeneity of devices, tasks have multiple running modes on nodes. Therefore, in addition to the mapping from task to node, coding should also include the running mode of the task. To further optimize the scheduling results, we also consider the execution order of tasks into coding, which is very beneficial for improving scheduling performance. Let's give a simple example. Assuming that both task *a* and task *b* are scheduled to the same node, and task *a* arrives before task *b*. Task *a* requires 8000MB of memory with a running time of 30 seconds, and task *b* requires 3000MB of memory with a running time of 20 seconds. The node currently has 5000MB of memory and after 20 seconds, an additional 5000MB of memory will be released. It is easy to calculate that the time required to complete both task *a* and task *b* is $20s + 30s + 20s = 70s$, while if task *b* runs before task *a*, the time required is $20s + 30s = 50s$. Therefore, it is also important to consider the execution order of tasks in coding.

Define the batch task to be scheduled as $N'' = \{n_1'', n_2'', \ldots, n_{|N''|}''\}$, the corresponding scheduling strategy is $G' = \{g_1', g_2', \ldots, g_{|N''|}'\}$, where $g_i' =< \xi_i', \zeta_i', \nu_i' >$, $1 \le i \le |N''|$, ξ_i', ζ_i' and ν_i' respectively represent node number, running mode, and waiting time, as mentioned in Section III. Therefore, the encoding of the particle is $code =< x_1, x_2, \ldots, x_{|N''|}, a_1, a_2, \ldots, a_p >$, which consists of two parts. The schematic diagram from encoding to scheduling strategy is as Fig. 3. The first part represents the node and running mode. For g_i', it can calculate nodes and running modes using the following equations:

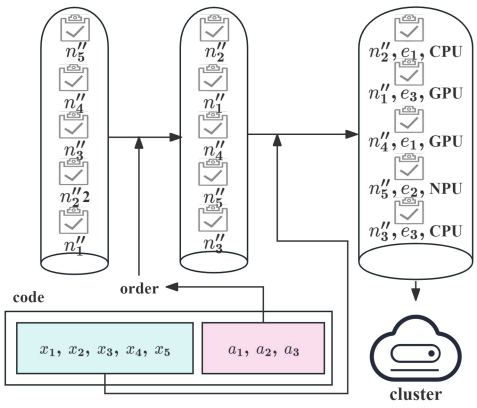

Fig. 3. Transformation from encoding to scheduling strategy.

$$\xi_i' = x_i \% K + 1 \tag{15}$$

$$\zeta_i' = \left\lfloor \frac{x_i}{K} \right\rfloor + 1 \tag{16}$$

ν_i' is the minimum waiting time that the *i*th task can run in the ζ_i'th mode on the ξ_i'th node, which means that when the node's resources are sufficient, the task does not need to wait. The second part is used to represent the scheduling order of tasks, which is decoded through the inverse operation of CE.

CE is an algorithm that maps a full permutation to an integer, and the integer mapped from a full permutation can also calculate the corresponding full permutation. This integer is essentially the lexicographic ranking of all permutations with the same length as it, defined as ϱ, and the length of the permutation is the number of tasks $|N''|$. Assuming that the order of the current tasks is represented by a full permutation as $(b_1, b_2, \ldots, b_{|N''|})$, then calculated by CE:

$$\varrho = \sum_{i=1}^{|N''|} \left(\sum_{j=i}^{|N''|} [b_i > b_j] \right) \left(|N''| - i\right)! \tag{17}$$

where $[b_i > b_j] = 1$ when $b_i > b_j$ and $[b_i > b_j] = 0$ when $b_i \le b_j$. Since ϱ represents the lexicographic ranking of a full permutation, and it starts counting from 0, $0 \le \varrho \le |N''|! - 1$. Our idea is to calculate ϱ through the part representing the tasks order in the code and obtain the order of the tasks from ϱ. This is the inverse operation of CE. The pseudo-code of the algorithm is as Algorithm 1.

Due to the maximum value of ϱ reaching $|N''|! - 1$, it is not appropriate to directly use ϱ as a dimension in the encoding. On the one hand, the large range of encoding values makes it difficult to search for the best results, and on the other hand, a small number of dimensions can also make the search difficult. Of course, a large number of dimensions is not conducive to search, as it can lead to a long search time. We use the n-base method to split the ϱ to obtain the

Algorithm 1: Inverse Operation of CE

Data: ϱ; $|N''|$.
Result: Permutation Ξ.
$\Xi = null$;
List $L = \{1, 2, 3, ..., |N''|\}$;
if $\varrho > |N''|! - 1$ **then**
  $\varrho = |N''|! - 1$;
for *i=1 to —N"—* **do**
  $index = \left\lfloor \frac{\varrho}{(|N''|-i)!} \right\rfloor$;
  Append $L[index]$ to Ξ;
  Delete the number $L[index]$ from L;
  Update ϱ to $\varrho\%(|N''| - i)!$;
return Ξ

appropriate number of dimensions and narrow the range of encoding values. Assuming the base number used is c, which means $a_i < c, 1 \leq i \leq p$, the equation is as follows:

$$\varrho = \sum_{i=1}^{p} c^{p-i} a_i \tag{18}$$

where p is the length of the second part of the *code*. If ϱ is greater than $|N''|! - 1$, let ϱ become $|N''|! - 1$. $c - 1$ is the maximum value of a_i, so the selection of c should be comprehensively determined based on the number of cluster nodes. Since the encoding needs to cover all permutations, satisfying $|N''|! \leq c^p$, the value of p can be obtained according to the following equation:

$$p = \left\lceil \log_c \left(|N''|! \right) \right\rceil \tag{19}$$

After completing the coding, we use PSO for solving optimization problems. It is easy to see that the distribution of the solution to the problem in the search space is discrete and irregular, which means that naive PSO may find it difficult to find the optimal solution. Therefore, we use PSO based on inertia weight optimization.

Many optimization methods for inertia weights are based in such a way: In the early stage of the search, the global search of PSO should be encouraged, which means increasing the speed of particles in the first few rounds of the search, as a high-speed is beneficial for particles to jump out of local optima, and reducing the speed of particles appropriately in the later stages of the search, as a small speed is beneficial for particles to find the optimal solution near the current solution, enabling the algorithm to converge at the end. In our modeling, the solution space is discrete and irregular, so it is necessary to encourage global search as much as possible, but for algorithm convergence, local search in the later stage is also required. We propose an inertia weight optimization method based on logistic function. We define the equation for the variation of inertia weight:

$$w = \frac{1}{1 + e^{-\frac{epoch\_num - epoch}{epoch\_num \cdot \lambda}}} \tag{20}$$

where *w* indicates the inertia weight, $epoch\_num$ indicates the total number of rounds, *epoch* indicates the number of rounds searched, λ is the descent coefficient, which meets $\lambda > 0$, representing the descent speed of w. The smaller the descent coefficient, the longer the global search process of PSO, but it may lead to non convergence of the algorithm. Therefore, an appropriate λ is helpful for the search of solutions.

Algorithm 2: Processing RNN Datasets and Using RNN for Batching Online Tasks and Scheduling

Data: Tasks submitted online; Task sequence for training RNN, i.e., the dataset.
Obtain the submitted task type and arrive time *s* from the dataset;
Obtain resource requirements of tasks *Q*, including all types of resources running in all modes on all nodes;
Calculate δ and *AR* for all tasks in the dataset;
Determine β and calculate *ST*, obtaining Λ;
Process Λ through dynamic programming and obtain the label of the dataset;
Design the structure of RNN and train it using the dataset, saving the trained model;
$\Gamma = \varnothing$;
$\tau' = +\infty$;
while *scheduling system startup* **do**
  if *Received a new task n at current time* τ **then**
    $\Gamma \cup \{n\}$;
    $\tau' = \min(\tau', RNN(< AR, \tau >))$;
    if $\tau' < \tau$ **then**
      Schedule all cached tasks in Γ;
      $\Gamma = \varnothing$;
      $\tau' = +\infty$;
    else
      Continue to wait;
  else
    Continue to wait;

The fitness function of PSO is designed based on Eq. (4), which is the weighted sum of TCT and TWT for this batch task. Therefore, PSO will optimize both TCT and TWT, and α determines the degree of dominance of TCT and TWT. When α is larger, TCT will be optimized preferentially. However, TCT and TWT are not always negatively correlated, which means that a larger α does not necessarily lead to a worse TWT. After the scheduling strategy is calculated, batch tasks are assigned to the corresponding nodes in the calculated order and nodes. At this point, the entire scheduling process is completed. The pseudocode for the entire process is as Algorithm 2.

V. Experiments

In this section, we discuss the practical application scenarios and effectiveness of the proposed EHRSM. In addition, we will also design comparative experiments to verify the superiority of the proposed method.

A. Establishment of Experimental Environment

EHRSM is mainly applied to EEC. To build such an EEC system, we have prepared cloud servers, edge servers, and end devices as nodes of the cluster. Among these devices,

all are equipped with CPU chips, but there are differences in architecture. In addition, some devices are equipped with GPU and NPU chips, so the resources of the cluster are heterogeneous.

To deploy and manage these devices, we use Kubernetes to build an EEC system. Kubernetes is an open-source container cluster management system that supports resource management and task dispatch functions for cluster nodes. However, Kubernetes does not support the monitoring of heterogeneous resources and can only obtain CPU-related information about nodes. Therefore, we have implemented a system ourselves to obtain heterogeneous resource information of nodes, and then use the proposed method for task scheduling. Finally, we deliver the task image and scheduling node ID together to Kubernetes for task dispatch.

For the tasks to be scheduled, we design three different types of computational tasks, namely AI-intensive, memory-intensive, and disk-intensive. Specifically, images of 5 AI-intensive tasks, 10 memory-intensive tasks, and 10 disk-intensive tasks are made. The AI-intensive task uses the MNIST dataset to train multiple CNN models and perform inference. Since neural network models can be accelerated using GPUs and NPUs, the AI-intensive task supports three running modes: CPU, GPU, and NPU. Memory-intensive tasks request a lot of memory for storing temporary data at runtime, so they have high demands on the CPU memory resources. Disk-intensive tasks use large files to perform frequent operations on the node's file system, thus placing a high demand on disk resources. These three types of tasks simulate the diversity of computational resource requirements in real-world scenarios.

Through the above environment construction, in our EEC system, the cluster has heterogeneous resources, and tasks can run in different modes on different devices with different running effects, which meets the required running environment for the proposed method.

B. Experimental Equipment Information

The experimental equipment consists of three parts, namely cloud servers, edge servers, and end devices. The specific hardware configurations of these devices are as follows:

Cloud server (x86): 1 server equipped with 8 Intel Xeon E5-2620 v4@2.10GHz CPUs (64G) and with 2 Nvidia Tesla T4 GPUs (16G); 1000GB disk capacity;

Edge server: 3 Atlas 200 DKs equipped with 2 A55 Arm core@1.6GHz CPUs (8G) with 2 Davinci AI core NPUs (8G); 50GB disk capacity;

End device: 4 Raspberry Pis equipped with 4 ARM cortex-A72@1.5GHz CPUs (8G) and with Broadcom VideoCore VI@500MHz (4G); 59GB disk capacity;

C. Experimental Design

In this section, we will provide a detailed explanation of the experimental process, showcasing the experimental data and the effectiveness of the proposed method.

1) Deployment Process: Firstly, we calculate the resource requirements of all tasks at different nodes and running modes, including CPU usage, CPU memory requirements, GPU usage, task runtime, and so on. For node resource requirements, as scheduling is real-time, the system will obtain the latest node information before each scheduling.

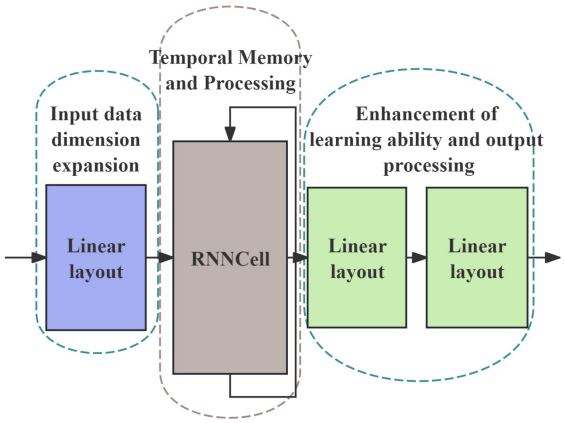

Fig. 4. The RNN structure used in our experiment.

In the proposed method, RNN training requires obtaining historical task submission records. We randomly generate 1000 tasks and arrive within 1000 seconds, as a submission task history record, which is used for model training. Because the scheduling of EEC requires real-time performance, the execution time of the scheduling algorithm should not be too long. For training the RNN with 1000 tasks, we set 100 schedulings to be able to schedule all the tasks, which represents an average number of tasks per batch of about 10. The RNN trained in this way adaptively adjusts the batch sizes of the tasks and does not deviate significantly from 10. The training dataset of RNN is processed using Eqs. (5)-(11), where $\beta = 0.05, |M| = 100$. Once the RNN training is complete, our method can be applied to the system for scheduling.

2) Method Parameters and Baseline Algorithms: In this section, the parameters of the proposed method will be provided in detail. At the same time, we will introduce several baseline algorithms for comparison, and their parameters will also be provided in detail.

EHRSM Firstly, we need to design a specific RNN structure that is required to have learning and predictive abilities. In our experiment, the RNN is designed with the structure shown in Fig. 4. The input dimension is a 1 * 2 vector $< AR, s >$. Firstly, preprocessing is performed to compress the data to [−2,2] for faster learning. We use a 2 * 4 linear layer to expand the input dimension, and then input it into RNNCell, including a 4 * 8 linear layer and an 8 * 8 linear layer. Finally, we use two linear layers 8 * 16 and 16 * 1 to enhance the learning and memory abilities of the model. The learning rate of the RNN is set to 0.05.

As for the setting of PSO, the base number c is set to 256, and the length p of order encoding will be calculated by Eq. (19). For example, when the number of batch tasks is 5, $p = 1$, and when the number of batch tasks is 10, $p = 3$. The particle number of PSO is set to 40, and the number of iterations is set to 50, which is a parameter setting that can ensure the algorithm can execute quickly.

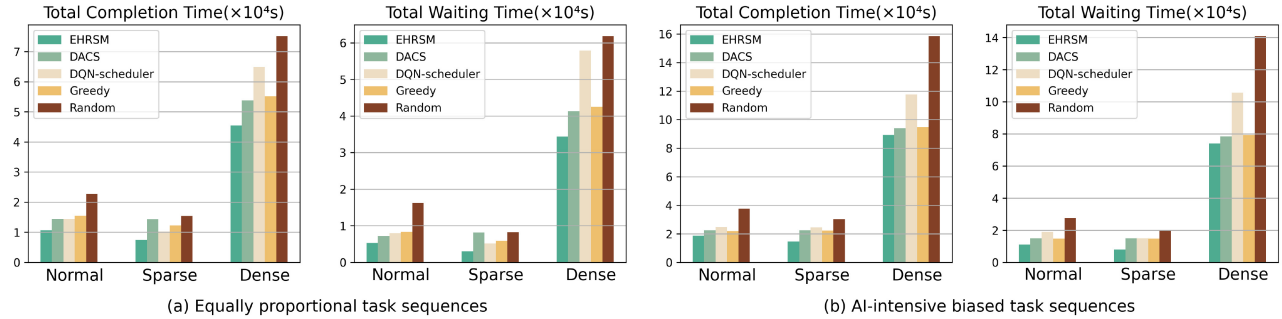

Fig. 5. The scheduling performance of different algorithms under different task sequences.

DACS [37] DACS was proposed to solve heterogeneous IoT node scheduling in edge computing. DACS improves the native scheduling algorithm of Kubernetes by adding processing latency and network latency to the optimization objective. When the latency of multiple nodes is similar to the optimal value, these nodes are further filtered to the final node based on the remaining resources. In this experiment, the delay of the nodes is calculated by Eq. (4).

DQN-scheduler [38] DQN-scheduler uses DQN-based reinforcement learning for job scheduling in Spark to reduce the completion time and cost of scheduling. The state space contains the heterogeneous resource information of the nodes and the task resource requirements, with a total of 106 dimensions. The action space includes the selection of different nodes and running modes, a total of 12 actions. During the training process, Eq. (4) is used to evaluate the model, and the best model parameters will be saved.

Greedy Greedy algorithm is a classic scheduling algorithm that makes the current optimal decision for each task without considering subsequent tasks.

Random Random algorithm selects nodes and running modes with equal probability for each task.

LJFP-PSO [32] LJFP-PSO is also an algorithm that uses PSO for scheduling. LJFP-PSO first preprocesses particles using the longest job to fastest processor (LJFP), and in addition, uses a binary matrix to represent the encoding of a particle. We set the number of particles to 40 and the number of iteration rounds to 50 to ensure the fairness of the baseline algorithm. According to the experimental setup in [32], the inertia weight uses a linear descent function, with a maximum value of 0.9, a minimum value of 0.2, and an acceleration coefficient of 2. All other settings are the same as those in [32].

GWO-GA [39] GWO-GA uses hybrid optimization methods to solve scheduling problems. This article is based on grey wolf optimization (GWO) and uses the crossover and mutation ideas of genetic algorithm (GA) to process the encoding, in order to improve the diversity of the population. The encoding method is the same as our proposed encoding, but there is no order encoding part. The number of groups is set to 40 and the number of iteration rounds is set to 50.

PSO-sigmoid [26] PSO-sigmoid uses inertia weight based on the sigmoid function to optimize PSO and applies it to scheduling. Compared with the logistic function we proposed, PSO-sigmoid has a shorter global search period. Its encoding method is the same as GWO-GA, and all other parameter settings related to PSO are the same as EHRSM.

Next, we will compare the performance of these algorithms.

3) Performance Experiments on the Online Scheduling System: To validate the performance of EHRSM on online scheduling, DACS, DQN-scheduler, Greedy and Random are used for comparison and different task sequences need to be designed. We design three kinds of task sequences, arriving at 100 tasks in 100s, arriving at 100 tasks in 200s, and arriving at 200 tasks in 100s, which are referred to as Normal, Sparse, and Dense task sequences, respectively, and are used to simulate the different tasks densities on the EEC scheduling system. In addition, we focus on whether the performance of the algorithm is significantly affected in scenarios with more heterogeneous tasks. Therefore, in terms of the types of tasks, we design equal proportional task sequences, which implies that the ratio of the number of AI-intensive, memory-intensive, and disk-intensive tasks is 1:1:1, and AI-intensive biased task sequences, which implies that the ratio of the number of AI-intensive, memory-intensive, and disk-intensive tasks is 3:1:1. Therefore, there are a total of six different task sequences, and in our experiments, we statistically measure the TCT and TWT of these algorithms. α in Eq. (4) is set to 0.5.

The experimental results are shown in Fig. 5. It can be intuitively seen that when the task sequence is dense, both TCT and TWT significantly increase. This is because more dense task sequences often lead to insufficient cluster resources, resulting in more tasks requiring long waiting times. It can be observed that EHRSM generally outperforms the baseline algorithm across task sequences. On the Equally proportional Sparse task sequence, EHRSM reduces TCT by 48.24%, 27.17%, 39.31%, and 51.68%, and TWT by 62.83%, 40.87%, 48.62%, and 63.20%, respectively, compared to DACS, DQN scheduler, Greedy, and Random, which demonstrates the ability of EHRSM to efficiently optimize the task completion time and waiting time simultaneously. DACS and Greedy differ only in that DACS further selects the node with the most remaining resources when multiple nodes have similar latencies, but due to the heterogeneity, this consideration is not always effective, and thus the performance of DACS and Greedy have similar performance. DQN-scheduler performs poorly in Dense task sequences, this is because the optimization strategy of DQN learns from historical task information, and the performance of DQN degrades when the real task sequences deviate from the training data. In addition,

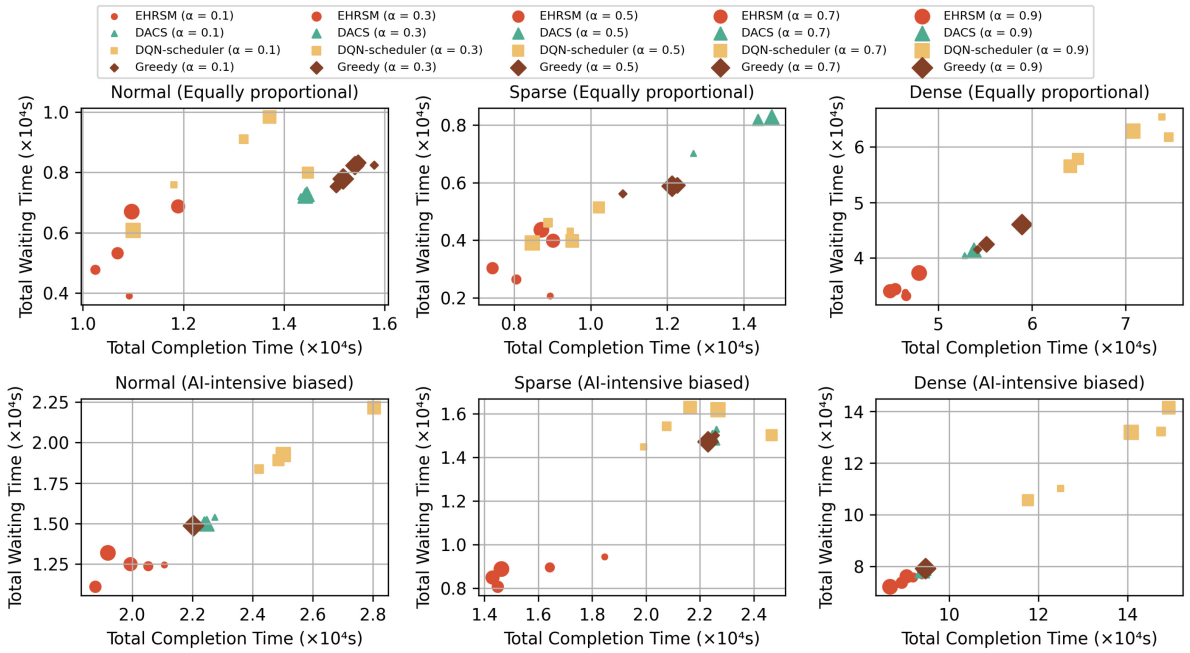

Fig. 6. The scheduling performance of different algorithms under different task sequences and α.

denser task sequences have a larger optimization space, which means that DQN's decision errors are magnified, resulting in DQN being inferior to DACS and Greedy.

To further observe the performance of these algorithms under different α, we set the α at 0.1, 0.3, 0.5, 0.7, 0.9 and validate them using different task sequences. The experimental results are shown in Fig. 6. Intuitively, the EHRSM is always in the lower left of the scatterplot, which is the position indicating the minimum of both TCT and TWT. Thus, EHRSM is further validated to have the best optimization capability compared to the baseline algorithm. Different α indicates that the optimization objective focuses on TCT and TWT differently, but experimentally, different α do not show this in the distribution, and only EHRSM show this distribution in Sparse (Equally proportional) and Normal (AI-intensive biased). This is mainly due to the inherent correlation between TCT and TWT, as TWT is a part of TCT, although minimizing TWT may lead to an increase in TCT due to neglecting task runtime.

4) Batch Task Scheduling Performance: In this section, we first compare the performance of EHRSM, LJFP-PSO, GWO-GA, and PSO-sigmoid in batch task scheduling. Then, we validate the performance improvement of our proposed EHRSM algorithm by utilizing CE and logistic function. Finally, we evaluate the algorithm execution time for single batch scheduling to discuss the feasibility of EHRSM. In the next experiments we keep $\alpha = 0.5$.

The scheduling performance of EHRSM is mainly derived from the optimization-seeking capability of the improved PSO. Therefore, the batch scheduling performance of EHRSM needs to be compared with other metaheuristic algorithms. We set up batch tasks of different lengths and count the TCT and TWT of a batch of tasks.

The experimental results are shown in Fig. 7. It can be seen that the TCT and TWT of EHRSM are always minimized among all the algorithms. When the batch size is 40, the TCT of EHRSM is reduced by 9.72%, 8.62%, and 5.45%, while the TWT is reduced by 40.97%, 36.97%, and 33.06%, respectively, compared with LJFP-PSO, GWO-GA, and PSO-sigmoid. This shows that the batch scheduling part of EHRSM also has good scheduling performance when the task batch is large. On the one hand, the sequential coding of EHRSM brings a larger optimization space, and on the other hand, the inertia weights of the logistic function enhance the optimization ability of PSO, which is the reason why the batch scheduling algorithm of EHRSM outperforms other metaheuristics.

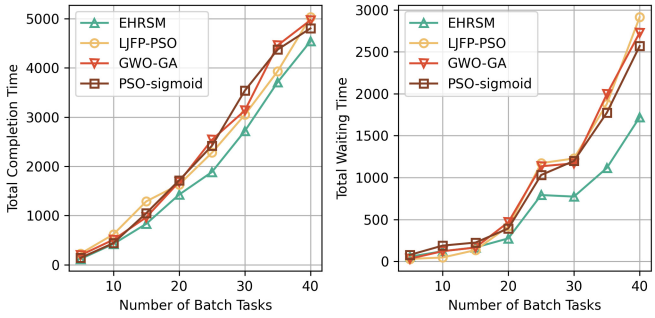

Fig. 7. Scheduling performance of various algorithms under different batch task lengths.

To better validate the effect of CE and logistic function inertia weights in EHRSM batch scheduling, we perform ablation experiments on these methods. The method that uses CE and logistic function is referred to as PSO&CE&W, the method that only uses CE is referred to as PSO&CE, the method that only uses logistic function is referred to as PSO&W, and the method that neither method uses is referred to as PSO. For the method with constant inertia weight, we set the inertia weight to 1. We validate their performance using batch tasks of different sizes.

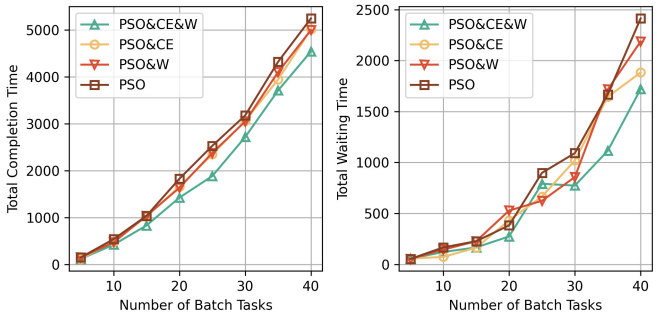

Fig. 8. The impact of CE and logistic function on algorithm performance.

TABLE I
THE EXECUTION SPEED OF THE BATCH SCHEDULING OF EHRSM UNDER DIFFERENT TASK BATCH LENGTHS

Batch Length	$T^c(s)$	TCT(s)	Proportion(T^c/TCT)
5	1.25	120.05	1.04%
10	2.78	423.95	0.66%
15	5.51	833.03	0.66%
20	8.40	1426.08	0.59%

The experimental results are shown in Fig. 8. It can be seen that PSO&CE&W outperforms the other three methods, and both PSO&CE and PSO&W outperform PSO. It indicates that both CE and inertia weight optimization enhance the performance of PSO and that these two optimizations are fusible due to their different optimization angles. Therefore, the efficient batch scheduling performance of PSO&CE&W ensures the effectiveness of EHRSM.

Finally, we evaluate the impact of the execution time of the batch scheduling to ensure whether it hinders the real-time performance of EEC scheduling. We set up different task lengths and count the execution time of the batch scheduling algorithm, denoted as T^c. The experimental results are shown in Table I. Obviously, the larger the batch of tasks leads to a longer execution time, but accordingly, the completion time of the batch is also larger. When the batch task length is 10, the execution time of the scheduling algorithm accounts for only 0.66% of the time, while when the batch task length is 20, although the scheduling algorithm executes for 8.40s, it only accounts for 0.59% of the completion time, which shows that the execution time of the batch scheduling algorithm of the EHRSM is fast enough to satisfy the real-time requirement.

5) The Effectiveness of RNN: In this section, we verify the effectiveness of batching online tasks using RNNs. We compare methods of batching using a number threshold and a time threshold, perform scheduling using the proposed batch scheduling algorithm, and ultimately compare the TCT and TWT of these batching methods. For the method of batching using a quantity threshold, all the arrived tasks are scheduled when the number of arriving tasks reaches a specified number. We set three quantities of 5, 10, and 15, denoted as 5n, 10n, and 15n. For the method of batching using a time threshold, all the arrived tasks are scheduled at fixed intervals. We set three intervals of 5s, 10s, and 15s, denoted as 5s, 10s, and 15s. In addition, we count the time spent on all tasks waiting for scheduling execution during the batching process, i.e., $T^w = \sum_{i=1}^{|N|} T_i^w$. We conduct experiments with different intensities and different proportions of task types, which are similar to the first experiment. α is set to 0.5.

TABLE II
PERFORMANCE COMPARISON OF RNN, NUMBER THRESHOLD, AND TIME THRESHOLD BATCHING METHODS FOR EQUALLY PROPORTIONAL TASK SEQUENCES AND THE DATA IS IN SECONDS

Models	Normal			Sparse			Dense		
	TCT	TWT	T^w	TCT	TWT	T^w	TCT	TWT	T^c
RNN	**10682**	**5329**	246	**7437**	**3040**	287	**45429**	34417	410
5n	10962	5697	**217**	8736	3724	423	46349	**34403**	**295**
10n	10768	5853	462	9239	4896	963	46832	35317	555
15n	11754	7039	717	9893	4712	1323	48985	37082	780
5s	10786	5385	287	8481	3415	**268**	46650	35874	635
10s	11284	6577	552	8550	3250	533	49465	36629	1185
15s	13551	7765	852	8081	3180	738	49767	37660	1670

TABLE III
PERFORMANCE COMPARISON OF RNN, NUMBER THRESHOLD, AND TIME THRESHOLD BATCHING METHODS FOR AI-INTENSIVE BIASED TASK SEQUENCES AND THE DATA IS IN SECONDS

Models	Normal			Sparse			Dense		
	TCT	TWT	T^w	TCT	TWT	T^w	TCT	TWT	T^c
RNN	**18769**	**11110**	280	**14483**	**8075**	351	89254	**73894**	490
5n	19519	11537	**263**	14847	8509	414	**88648**	74184	**307**
10n	20241	12800	483	18298	10798	874	91805	77109	532
15n	22398	14066	718	20947	13195	1314	94067	77270	777
5s	20010	11863	263	14487	8125	**289**	91665	75936	642
10s	21398	13611	543	16175	8803	544	98063	80886	1112
15s	22735	15994	858	17580	10477	789	99154	83325	1627

The experimental results are shown in Tables II, III. From the tables, it can be seen that no matter which kind of task sequence, the performance of RNN is better than the other methods in the vast majority of cases, which is since RNN can adaptively adjust the number of tasks in a batch according to the characteristics of the tasks. Since the arrival time of the tasks is random, even with a Normal task sequence, there will be some periods when tasks arrive in bursts and some periods when tasks arrive sparsely. In this case, the adaptation of RNN is crucial. Different batching strategies work differently when the densities of the tasks change drastically. It shows that the RNN is indeed effective in batching the tasks and transforming online task scheduling into offline scheduling.

VI. CONCLUSION

In this paper, an EEC heterogeneous resources scheduling method based on RNN and PSO is proposed. The tasks published online are cached using an RNN trained from a dataset processed through dynamic programming, transforming online task scheduling into offline task scheduling. The cached batch tasks use PSO optimized by CE for encoding to schedule heterogeneous resources in the EEC, thereby achieving efficient task scheduling and improving the quality of cloud services. The experiment has proven that compared with some mainstream scheduling algorithms, the proposed method has more efficient scheduling performance, can greatly reduce task waiting time, improve response speed, and thus improve service quality. In the experiment, we also found some

areas worth improving. As a heuristic search algorithm, PSO requires the support of computing resources for its operation. Therefore, in different scenarios, an appropriate number of particles and search rounds is beneficial for saving computing resources and improving computing speed. In addition, in practical application scenarios, tasks may have characteristics such as priority or dependency relationships, in which case more complex modeling is needed to solve the problem. In future work, we will further address and research these issues.

REFERENCES

[1] Z. Yang, B. Liang, and W. Ji, "An intelligent end–edge–cloud architecture for visual IoT-assisted healthcare systems," *IEEE Internet Things J.*, vol. 8, no. 23, pp. 16779–16786, Dec. 2021.

[2] G. Liu et al., "An adaptive DNN inference acceleration framework with end–edge–cloud collaborative computing," *Future Gener. Comput. Syst.*, vol. 140, pp. 422–435, Mar. 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167739X22003570

[3] C. Ding, F. Ding, S. Gorbachev, D. Yue, and D. Zhang, "A learnable end-edge-cloud cooperative network for driving emotion sensing," *Comput. Elect. Eng.*, vol. 103, Oct. 2022, Art. no. 108378. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S004579062200595X

[4] S. Duan et al., "Distributed artificial intelligence empowered by end-edge-cloud computing: A survey," *IEEE Commun. Surveys Tuts.*, vol. 25, no. 1, pp. 591–624, 1st Quart., 2023.

[5] T. Bezdan, M. Zivkovic, N. Bacanin, I. Strumberger, E. Tuba, and M. Tuba, "Multi-objective task scheduling in cloud computing environment by hybridized bat algorithm," *J. Intell. Fuzzy Syst., Appl. Eng. Technol.*, vol. 42, no. 1, pp. 411–423, 2022.

[6] S. Wang, T. Zhao, and S. Pang, "Task scheduling algorithm based on improved firework algorithm in fog computing," *IEEE Access*, vol. 8, pp. 32385–32394, 2020.

[7] B. Jana, M. Chakraborty, and T. Mandal, "A task scheduling technique based on particle swarm optimization algorithm in cloud environment," in *Proc. SoCTA*, 2019, pp. 525–536, doi: 10.1007/978-981-13-0589-4_49.

[8] M. H. Shirvani and R. N. Talouki, "A novel hybrid heuristic-based list scheduling algorithm in heterogeneous cloud computing environment for makspan optimization," *Parallel Comput.*, vol. 108, Dec. 2021, Art. no. 102828.

[9] S. Gupta et al., "Efficient prioritization and processor selection schemes for HEFT algorithm: A makespan optimizer for task scheduling in cloud environment," *Electronics*, vol. 11, no. 16, p. 2557, 2022. [Online]. Available: https://www.mdpi.com/2079-9292/11/16/2557

[10] B. Jamil, M. Shojafar, I. Ahmed, A. Ullah, K. Munir, and H. Ijaz, "A job scheduling algorithm for delay and performance optimization in fog computing," *Concurr. Comput., Pract. Exp.*, vol. 32, no. 7, 2020, Art. no. e5581. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5581

[11] F. Ebadifard and S. M. Babamir, "Autonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment," *Clust. Comput.*, vol. 24, pp. 1075–1101, Jun. 2021.

[12] F. Ebadifard, S. M. Babamir, and S. Barani, "A dynamic task scheduling algorithm improved by load balancing in cloud computing," in *Proc. 6th Int. Conf. Web Res. (ICWR)*, 2020, pp. 177–183.

[13] S. Nabi, M. Ibrahim, and J. M. Jimenez, "DRALBA: Dynamic and resource aware load balanced scheduling approach for cloud computing," *IEEE Access*, vol. 9, pp. 61283–61297, 2021.

[14] N. P. S. Kumar, S. Arjun, B. Dhivya, and S. K. S. Sri, "Determining energy consumption in heterogenous cloud computing by usage of RMRECFS workflow with cost confinement," in *Proc. 4th Int. Conf. Invent. Res. Comput. Appl. (ICIRCA)*, 2022, pp. 1287–1292.

[15] T. Long et al., "A novel fault-tolerant scheduling approach for collaborative workflows in an edge-IoT environment," *Digit. Commun. Netw.*, vol. 8, no. 6, pp. 911–922, 2022.

[16] S. Souravlas and S. Katsavounis, "Scheduling fair resource allocation policies for cloud computing through flow control," *Electronics*, vol. 8, no. 11, p. 1348, 2019. [Online]. Available: https://www.mdpi.com/2079-9292/8/11/1348

[17] M. Hosseini Shirvani and R. Noorian Talouki, "A novel hybrid heuristic-based list scheduling algorithm in heterogeneous cloud computing environment for makespan optimization," *Parallel Comput.*, vol. 108, Dec. 2021, Art. no. 102828. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167819121000764

[18] A. Kaur, P. Singh, R. Singh Batth, and C. Peng Lim, "Deep-Q learning-based heterogeneous earliest finish time scheduling algorithm for scientific workflows in cloud," *Softw., Pract. Exp.*, vol. 52, no. 3, pp. 689–709, 2022. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/spe.2802

[19] K. Li, "Design and analysis of heuristic algorithms for energy-constrained task scheduling with device-edge-cloud fusion," *IEEE Trans. Sustain. Comput.*, vol. 8, no. 2, pp. 208–221, Apr.–Jun. 2023.

[20] V. P. Verma, N. S. Naik, and S. Kumar, "Reinforcement learning based scheduling for spark jobs in cloud environment," in *Proc. IEEE 9th Uttar Pradesh Sect. Int. Conf. Elect., Electron. Comput. Eng. (UPCON)*, 2022, pp. 1–6.

[21] A. Lakhan and X. Li, "Content aware task scheduling framework for mobile Workflow applications in heterogeneous mobile-edge-cloud paradigms: CATSA framework," in *Proc. IEEE Int. Conf Parallel Distrib. Process. Appl., Big Data Cloud Comput., Sustain. Comput. Commun., Soc. Comput. Netw. (ISPA/BDCloud/SocialCom/SustainCom)*, 2019, pp. 242–249.

[22] T. Oo and Y.-B. Ko, "Application-aware task scheduling in heterogeneous edge cloud," in *Proc. Int. Conf. Inf. Commun. Technol. Converg. (ICTC)*, 2019, pp. 1316–1320.

[23] H. Wang, "Intelligent scheduling strategy for heterogeneous multi-terminal access tasks for complex cloud-edge collaborative computing," in *Proc. 4th Int. Conf. Inf. Sci., Parallel Distrib. Syst. (ISPDS)*, 2023, pp. 412–417.

[24] G. M. Nayeem, M. Fan, and Y. Akhter, "A time-varying adaptive inertia weight based modified PSO algorithm for UAV path planning," in *Proc. 2nd Int. Conf. Robot., Elect. Signal Process. Techn. (ICREST)*, 2021, pp. 573–576.

[25] M. Li, H. Chen, X. Wang, N. Zhong, and S. Lu, "An improved particle swarm optimization algorithm with adaptive inertia weights," *Int. J. of Inf. Technol. Decis. Making*, vol. 18, no. 03, pp. 833–866, 2019. [Online]. Available: https://doi.org/10.1142/S0219622019500147

[26] X. Huang, C. Li, H. Chen, and D. An, "Task scheduling in cloud computing using particle swarm optimization with time varying inertia weight strategies," *Clust. Comput.*, vol. 23, no. 2, pp. 1137–1147, 2020.

[27] N. Kardani, A. Bardhan, P. Samui, M. Nazem, P. G. Asteris, and A. Zhou, "Predicting the thermal conductivity of soils using integrated approach of ANN and PSO with adaptive and time-varying acceleration coefficients," *Int. J. Thermal Sci.*, vol. 173, Mar. 2022, Art. no. 107427. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1290072921005822

[28] R. Li, Q. Zhuang, N. Yu, R. Li, and H. Zhang, "Improved hybrid particle swarm Optimizer with sine-cosine acceleration coefficients for transient electromagnetic inversion," *Curr. Bioinf.*, vol. 17, no. 1, pp. 60–76, 2022.

[29] W. Yang, X. Zhou, and Y. Luo, "Simultaneously optimizing inertia weight and acceleration coefficients via introducing new functions into PSO algorithm," *J. Phys., Conf. Ser.*, vol. 1754, no. 1, Feb. 2021, Art. no. 12195. [Online]. Available: https://dx.doi.org/10.1088/1742-6596/1754/1/012195

[30] A. T. Kiani et al., "An improved particle swarm optimization with chaotic inertia weight and acceleration coefficients for optimal extraction of PV models parameters," *Energies*, vol. 14, no. 11, p. 2980, 2021. [Online]. Available: https://www.mdpi.com/1996-1073/14/11/2980

[31] Y. Du and F. Xu, "A hybrid multi-step probability selection particle swarm optimization with dynamic chaotic inertial weight and acceleration coefficients for numerical function optimization," *Symmetry*, vol. 12, no. 6, p. 922, 2020. [Online]. Available: https://www.mdpi.com/2073-8994/12/6/922

[32] S. A. Alsaidy, A. D. Abbood, and M. A. Sahib, "Heuristic initialization of PSO task scheduling algorithm in cloud computing," *J. King Saud Univ., Comput. Inf. Sci.*, vol. 34, no. 6, pp. 2370–2382, 2022. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1319157820305279

[33] R. M. A. N. I. J. Abdullayeva, "PSO-based load balancing method in cloud computing," *Autom. Control Comput. Sci.*, vol. 53, no. 1, pp. 45–55, 2019.

[34] A. M. Senthil Kumar, P. Krishnamoorthy, S. Soubraylu, J. K. Venugopal, and K. Marimuthu, "An efficient task scheduling using GWO-PSO algorithm in a cloud computing environment," in *Proc. Int. Conf. Intell. Comput., Inf. Control Syst.*, pp. 751–761.

[35] Richa and B. N. Keshavamurthy, "Improved PSO for task scheduling in cloud computing," in *Evolution in Computational Intelligence: Frontiers in Intelligent Computing: Theory and Applications*, vol. 1, V. Bhateja, S.-L. Peng, S. C. Satapathy, and Y.-D. Zhang, Eds., Singapore: Springer, 2021, pp. 467–474, doi: 10.1007/978-981-15-5788-0_45.
[36] M. S. A. Khan and R. Santhosh, "Task scheduling in cloud computing using hybrid optimization algorithm," *Soft Comput.*, vol. 26, pp. 13069–1307, Dec. 2022.
[37] W.-K. Lai, Y.-C. Wang, and S.-C. Wei, "Delay-aware container scheduling in Kubernetes," *IEEE Internet Things J.*, vol. 10, no. 13, pp. 11813–11824, Jul. 2023.
[38] M. T. Islam, S. Karunasekera, and R. Buyya, "Performance and cost-efficient spark job scheduling based on deep reinforcement learning in cloud computing environments," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 7, pp. 1695–1710, Jul. 2022.
[39] I. Behera and S. Sobhanayak, "Task scheduling optimization in heterogeneous cloud computing environments: A hybrid GA-GWO approach," *J. Parallel Distrib. Comput.*, vol. 183, Jan. 2024, Art. no. 104766. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0743731523001363

Haijie Wu is currently pursuing the M.S. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, supervised by Dr. W. Lin. His research interests mainly include cloud edge collaboration, edge computing, and AI algorithms.

Wangbo Shen received the B.S. degrees from Changsha University, Changsha, China, in 2012 and 2016, respectively, and the M.S. degrees from Central South University, Changsha, in 2016 and 2019, respectively. He is currently pursuing the Ph.D. degree with the School of Computer Science and Engineering, South China University of Technology, Guangzhou, China, supervised by Dr. W. Lin. His research interests mainly include Kernel learning, AutoML, and edge computing.

Weiwei Lin (Senior Member, IEEE) received the B.S. and M.S. degrees from Nanchang University in 2001 and 2004, respectively, and the Ph.D. degree in computer application from the South China University of Technology in 2007, where he is currently a Professor with the School of Computer Science and Engineering. He was a Visiting Scholar with Clemson University from 2016 to 2017. He has published more than 150 papers in refereed journals and conference proceedings. His research interests include distributed systems, cloud computing, and AI application technologies. He has been a Reviewer for many international journals, including the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON COMPUTERS, and the IEEE TRANSACTIONS ON CYBERNETICS. He is a Distinguished Member of CCF.

Wei Li (Senior Member, IEEE) received the Ph.D. degree from the School of Information Technologies, The University of Sydney, where he is currently an ARC DECRA Fellow with the Centre for Distributed and High Performance Computing, School of Computer Science. His research interests include edge computing, sustainable computing, task scheduling, energy efficiency, and the Internet of Things. He is the recipient of four IEEE or ACM conference best paper awards. He received the IEEE TCSC Award for Excellence in Scalable Computing for Early Career Researchers in 2018 and the IEEE Outstanding Leadership Award in 2018. He is a Senior Member of IEEE Computer Society and a member of ACM.

Keqin Li (Fellow, IEEE) is a SUNY Distinguished Professor of computer science with the State University of New York. He is also a National Distinguished Professor with Hunan University, China. His current research interests include, fog computing and mobile edge computing, energy-efficient computing and communication, embedded systems and cyber-physical systems, heterogeneous computing systems, big data computing, high performance computing, computer architectures and systems, computer networking, ML, and intelligent and soft computing. He is currently an Associate Editor of the *ACM Computing Surveys* and the *CCF Transactions on High Performance Computing*. He has served an Editorial Boards of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, the IEEE TRANSACTIONS ON COMPUTERS, the IEEE TRANSACTIONS ON CLOUD COMPUTING, the IEEE TRANSACTIONS ON SERVICES COMPUTING, and the IEEE TRANSACTIONS ON SUSTAINABLE COMPUTING. He is an AAIA Fellow. He is also a member of Academia Europaea (Academician of the Academy of Europe).